计算机 C/C++语言系列丛书

# C语言保护方式编程捷径

李增泰　编著

学苑出版社

## 希望汉字系统
## UCDOS 3.1

1985年，北京希望电脑公司首次推出汉字系统UCDOS 1.0，其优越的性能，得到了广大微机用户的关心、爱护和支持，在此，我们表示忠心的感激！

随着时间的推移，UCDOS不断地追求汉字系统的最完美境界，终于以最强的功能、最好的兼容性和最惠的价格，遥遥领先于其它汉字系统，成为被追赶的目标。

作为UCDOS家族的最新成员，UCDOS 3.1必将成为汉字系统领域中最耀眼的明珠。

### 最新奉献

- 首家提供自然码最新5.20版，提供五笔、普通、全拼、简拼、双拼，支持联想输入
- 显示驱动与硬件无关，可支持任意显示设备及任意分辨率和任意种显示颜色下显示汉字、输入汉字等，根据公开接口，各厂商都可以为UCDOS 3.1写显示驱动程序。如：CGA、EGA、VGA、HGC、TVGA、PVGA等显示设备；支持高速滚屏（硬件）
- 支持大量西文软件直接运行，使用智能的直接写屏和西文制表符识别算法，如实地再现了原版软件的神奇风采
- 支持256色下汉字的正常输入、输出及直接写屏，可在任何扩展图形模式下进行。提供任意尺寸和颜色的矢量汉字显示功能及大量屏幕作图功能
- 全面支持WPS，WPS可在任意高版本DOS和网络环境下运行，并可同时使用UCDOS提供的26种矢量字库，模拟显示和打印速度提高2～3倍
- 支持所有打印机，打印精度最高可达1200dpi，西文制表符打印识别，完美的彩色打印，利用HP打印机特有的数据压缩功能打印，令打印速度提高一至二倍
- 新的Readme，可随意跳转。更完整的手册， 为用户及开发人员提供大量方便

UCDOS产品价格如下：

| | | |
|---|---|---|
| UCDOS 3.1 单用户版 | : | 980.00 元 |
| UCDOS 3.1 网络版 | : | 2200.00 元 |
| UCDOS 3.0 单用户版升级 | : | 100.00 元 |
| UCDOS 3.0 网络版升级 | : | 200.00 元 |
| UCDOS 3.0 以下单用户升级 | : | 400.00 元 |
| UCDOS 3.0 以下网络升级 | : | 800.00 元 |

注 ： 代理价格面议

好消息：凡其他汉字系统的正式用户皆可凭用户卡，按半价交叉升级到UCDOS 3.1版！

诚征全国行销伙伴　　共同开拓软件市场

希望软件热线：　8422024　8422025

地　址：北京海淀路82号希望公司软件部
电　话：01-2579873，2579826，8422024，8422025
联系人：周东，张军，夏克
传　真：01-8422024，2561057

收款单位：北京希望电脑公司软件部
开 户 行：工商行海淀分理处
帐　　号：661924-61
邮　　编：100080



ISBN 7-5077-0875-6

9 787507 708752 >

计算机 C/C⁺⁺语言系列丛书

ISBN7—5077—0875—6/TP・24

本册定价：17.00元

计算机 C/C++语言系列丛书

# C 语言保护方式编程捷径

李增泰　编著

学 苑 出 版 社
1994.

(京)新登字151号

## 内容提要

当今，计算机CPU性能不断翻新，而软件开发却略显迟滞。例如微处理器80386、80486崭新而卓越的工作方式——"保护方式"，其性能就远远没有得到充分发挥，因为好多程序还只是局限于386对8086的兼容方式——"实方式"之下进行操作。本书则力图打破这种局面，向读者介绍"保护方式"下C语言程序的开发，以及80386汇编语言的使用。

**欲购本书的用户，请直接与北京8721信箱联系，电话：2562329，邮码：100080。**

计算机C/C++语言系列丛书

**C语言保护方式编程捷径**

---

**编　　著**：李增泰
**责任编辑**：甄国宪
**出版发行**：学苑出版社　　**邮政编码**：100036
**社　　址**：北京市海淀区万寿路西街11号
**印　　刷**：施园印刷厂
**开　　本**：787×1092　1/16
**印　　张**：14.875　　**字　数**：342千字
**印　　数**：1～5000册
**版　　次**：1994年6月北京第1版第1次
ISBN7－5077－0875－6/TP・24
**本册定价**：17.00　元

---

学苑版图书印、装错误可随时退换

# 前　言

## 一．内容介绍

使用C语言编写保护方式程序，便是本书向读者讲述的内容。另外书中还介绍了一些相关的技术，如80386CPU的工作方式、以及DOS-Extender的必备知识。

80386的保护方式

80386CPU最关键的进步就在于改变了存贮管理思想，并丰富了各种工作方式，它们包括实地址方式、保护方式、虚拟实方式。而对于一般计算机人员来说，最熟悉的莫过于实方式，因为DOS一直都是一个实地址方式的操作系统，而从未在保护方式方面作过什么设计。

80386本身的数据宽度是32位，直接寻址能达到4GB，而实方式所依据的是16位数据宽度，直接寻址仅仅有1MB。若要充分发挥386CPU的高级特性，那只有选择保护方式。而且80386的先进的存贮管理，作为并行多任务的根据，只有在保护方式下才能得以体现。

DOS-Extender

DOS-Extender属于DOS下的一个实用程序，它为DOS扩展了大量的保护方式功能，使系统能够支持保护方式程序的开发和运行。

C语言保护方式编程

C语言是当今最最流行的编程语言，但用它直接编写保护方式程序却是一个新鲜的领域。本书就将介绍这一技术，使众多C程序员的工作迅速进入一个更高的境界。

## 二．本书的意图

用C语言编写保护方式程序，尚属崭新的技术。以前386或386以上微机，其CPU的卓越性能纵然为大家所了解，但由于缺乏必要的手段和工具，也只能望洋兴叹。

微机从8088XT、80286AT到80386、80486的进步，让计算机用户直接体会到的只有运行速度的提高，而其最具突破性的进步是CPU工作方式的改变，这一点却不能为大家随意驾驭，甚至大寻址空间的好处都不能方便地在自己的程序中得以体现。

与此技术相关的软件和资料在国内市场已经出现，例如Watcom C、DOS-Extender，但

是它们所引起的注意，却远远没能达到其理应达到的程度。关键在于保护方式还没有被普遍了解，保护方式编程技术更未被一般计算机人员所掌握。

《C语言保护方式编程捷径》意欲为改变这一状况提供必要的动力，帮助您尽快走进保护方式编程领域，使386、486的卓越性能为您尽情享用。

# 目　录


**第一章　80386 工作方式** …… 1
　第一节　80386 的设计特色 …… 1
　第二节　80386 有哪些工作方式 …… 1
**第二章　保护方式下的内存管理** …… 3
　第一节　内存管理模式 …… 3
　第二节　内存分配 …… 4
**第三章　了解 DOS-EXTENDER** …… 6
　第一节　入门者须知 …… 6
　第二节　DOS-EXTENDER 使用简述 …… 7
**第四章　术语** …… 12
**第五章　在保护方式下编程** …… 14
　第一节　关于 32 位 80386 C 的编译 …… 15
　第二节　关于 32 位 80386 汇编 …… 18
　第三节　DOS-EXTENDER …… 20
　第四节　设计保护方式程序库 …… 21
**第六章　编写并运行保护方式的程序** …… 23
　第一节　用 C 语言编写保护方式程序 …… 23
　第二节　C 程序调用 80386 汇编子程序 …… 28
**第七章　库函数头文件** …… 44
　第一节　函数预定义文件 TPROTO. H …… 44
　第二节　结构定义文件 TSTRUCT. H …… 59
　第三节　键盘定义文件 KEYBOARD. H …… 64
　第四节　ASCII 码定义文件 ASCII. H …… 72
**第八章　库函数遍览** …… 75
　第一节　光标控制部分 …… 76
　第二节　屏幕控制部分 …… 86
　第三节　发声控制部分 …… 106
　第四节　键盘控制部分 …… 117
　第五节　矩形块管理部分 …… 131
　第六节　窗口管理部分 …… 146
　第七节　鼠标管理部分 …… 161

**第九章　库函数源代码清单**……………………………………………… 167
第一节　按所属章节排序索引………………………………………… 167
第二节　按名称字典排序索引………………………………………… 169
第三节　源代码清单…………………………………………………… 171

# 第一章　80386 工作方式

本章内容提要

- 80386 芯片的设计特色
- 80386 工作方式探讨

不管您是否熟悉 80386 的细节，在学习用 C 语言编写保护方式程序之前，了解一下 80386 的工作方式总是件受益匪浅的事情。

## 第一节　80386 的设计特色

80386 微处理器的设计有很多特色，如：并行多任务、大寻址空间、软件保护、分页虚存、以及与早期软硬件的向上兼容。

80386 的内部寄存器与数据通路，宽度都是 32 位，有效地址也是 32 位，因此能一次处理或传输 32 位的数据，并提供高达 4GB 的线性地址空间。

80386 包括多个密切配合的功能单元，如：指令预先提取单元、指令译码单元、执行单元、分段单元、分页单元、总线接口单元。在一条指令被执行的同时，其它指令被预先提取、译码，而且芯片上的地址计算和分页换算也都同时进行。多个单元的同时动作，正是多任务并行处理的基础。

80386 的分页单元负责存贮器管理、存放地址转换数据，并且提供虚拟存贮器能力，使得程序可以操作的地址空间比芯片的实际容量大得多。

由于完善的向上兼容特性，16 位机的实方式属性可以作为 80386 的一个子功能表现出来。

80386 的设计还有许多过人之处，留作此书以外的话题，供您研究。

## 第二节　80386 有哪些工作方式

80386 拥有完整的 32 位运算能力，同时具备作为 16 位 8086 以及 80286 的功能。其丰富的工作方式使得 80386 的计算能力极为强大且富有弹性。工作方式有如下三种：

- REAL，　　　　实地址方式
- PROTECTED，　保护方式
- VIRTUAL8086，　虚拟 8086 方式

三种方式的寻址方法不同、寻址空间大小不同。

在 PC 机的发展过程中，随着 CPU 的进步，其工作方式也经历了几次关键性的变化。

· 16 位的 8086/8088：只有实地址方式，寻址范围仅仅 1MB

· 16 位的 80286：加进了高级的 16 位保护方式，寻址存储器高达 16MB

· 80386：在保护方式中又加进了 32 位运算，寻址空间大于 4GB，并且添加了虚拟 8086 方式，以兼容现有的 8086 软件

这一发展过程可参见图 1—2—1

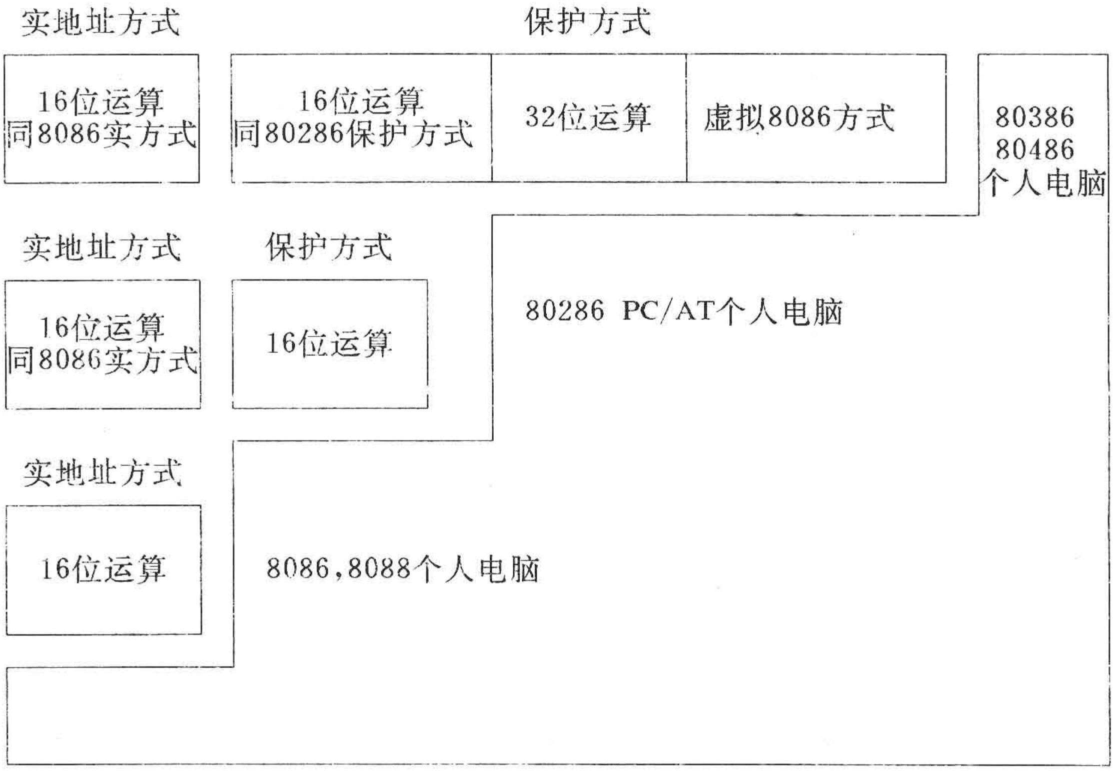


图1—2—1　CPU及相关个人电脑工作方式发展过程

保护方式是 80386 工作方式中非常优秀的一种，不仅具有 32 位的数据传输与寻址能力，而且能引用具有分页作用的虚拟存储器，具有坚强的软件安全防范措施。保护方式编程已经成为未来软件开发中一个不可忽视的方向。

80386 的众多先进特性只有在保护方式下才能得到充分体现。在保护方式下，可以构造一个多任务环境，任务之间是相互隔绝的，任务和操作系统之间也是隔绝和保护的。同时任务之间可以相互切换，8086 实方式程序就可以作为一个任务，运行于保护方式划分出来的一块空间之中，而且与其它任务之间互不影响。

80386 的硬件在保护方式下，提供完全的 32 位运算。不管是 16 位还是 32 位运算，只要在 80386 保护方式下，就能启动其分页单元，以完全支援虚拟存储器，使编程人员免于为实地址的寻址极限大伤脑筋。

实方式已经为大家所明了，可保护方式的情况也许不同，尤其对使用高级语言的编程人员，诸如 C 语言。如何轻而易举地充分发挥计算机硬件的先进性能，如何追随软件的时代潮流，希望这里能给您提供一个欣喜的开端。

# 第二章　保护方式下的内存管理

在这一章，除介绍了保护方式下的内存管理以外，还提到了 DOS-EXTENDER，它实际上是介于 DOS 和应用程序之间的一个界面，用来支持 32 位程序的运行和开发。关于 DOS-EXTENDER 在第三章将集中讨论。

## 第一节　内存管理模式

80386、80486 通过其分页、分段功能实现内存管理。

### 2.1.1　分段

保护方式下的寻址空间，同样被分成若干个“段”，只是各段之间相互隔绝，不可随意访问。如果一个段中的程序企图引用其它段的数据，将产生严重故障。

每个段都有自己专用的描述符，定义本段的 32 位线性基地址、边界、属性、和权限。

所有段的段描述符罗列在一起，组成一个描述符序列，称为描述符表。

当然表中的每一项(一个描述符)都被分配一个序号，这个序号被叫做段选择器。

程序如何控制、查阅分段信息呢?

程序先从段寄存器(CS)中取出段选择器，再根据段选择器值从描述符表中取出段描述符，最后从段描述符获得分段的各种信息。

### 2.1.2　分页

把 32 位的线性地址转换到 32 位的物理地址，即相应的地址总线上，就是通过 80386 的分页机制。

线性地址中有 20 位用来选择内存的物理页面，而另 12 位描述页面内的偏移量。一个物理页面大小是 4KB。

### 2.1.3　内存访问

80386 通过一个段选择器和一个段内偏移量来索引到内存。保护方式下的内存访问，利用如下方法构造物理地址：

- 从段描述符得到段的线性基地址
- 把段内偏移量与基地址相加，得到一个 4GB 空间内的线性地址空间
- 通过分页机制把线性地址转换成和地址总线相对应的物理地址

## 第二节　内存分配

### 2.2.1　分配内存的方法

内存包括常规内存、扩展内存。

常规内存，即 1MB 以内的存贮空间。

扩展内存，即 1MB 以上的所有存贮空间。

应用程序有权使用常规内存，也有权使用扩展内存。但通常是常规内存优先分配，然后才分配扩展内存。

内存分配和重分配总是以页为单位。内存分配没有特殊的顺序，因为不再需要的内存一旦被释放，任何后继的内存请求都可获得其使用权。

关于扩展内存的分配，有三种方法：

· 直接扩展内存，利用 INT15H 的 88 号功能调用，获取扩展内存的大小，接管并保留对内存的控制

· VCPI 虚拟控制程序接口内存，系统引导时，由 EMS 仿真程序截取一块内存，DOS-EXTENDER 利用 VCPI 动态地分配来自这块内存的页面

· XMS 内存，使用 HIMEM. SYS 分配扩展内存

DOS-EXTENDER 支持上述所有三种方法，但注意，同一时刻只能让一种方法有效，如若 VCPI 和 XMS 同时存在，则 DOS-EXTENDER 使用 VCPI，而非 XMS。

### 2.2.2　常规内存的使用

一般情况下，DOS-EXTENDER 占用大约 80KB(至少 65KB)常规内存归自己使用，并且通过 MS-DOS 调用给应用程序内存缓冲区保留一块内存，因为常规内存是可以从 MS-DOS 获得的内存。常规内存分配情况参见图 2—2—2—1

### 2.2.3　直接扩展内存的使用

直接扩展内存，指 1MB 以上尚未被分配出去的内存。

初始化期间，DOS-EXTENDER 检查有多少直接扩展内存还没被其它程序所占用，然后再给予分配。一般有两种分配策略。

DOS-EXTENDER 以页为单位、自上而下分配扩展内存，直到用完为止。假设一个保护模式主程序含有若干个保护模式子程序，主程序一旦装入运行其子程序，子程序就可以使用主程序的扩展内存，子程序结束后，把内存自动释放并归还给主程序。

扩展内存页面被释放时，DOS-EXTENDER 将其标记下来，以备后继程序使用。

图2—2—1 常规内存分配表

# 第三章　DOS-EXTENDER 了解

DOS-EXTENDER 是在 DOS(MS-DOS 或 PC-DOS)下运行或开发 32 位应用程序的保护模式环境，在其中运行的程序能直接访问机器的所有内存。本书将集中探讨的内容—C 语言编写保护方式程序，就将完全依赖这一环境。

DOS 一直都只是一个实方式的操作系统，而借助于 DOS-EXTENDER，我们直接感受到的将是 386 以上微处理器完善的保护方式性能。

## 第一节　入门者须知

DOS-EXTENDER 有哪些版本？哪些相关产品？适用于什么环境？兼容性如何？下面逐一介绍。

### 3.1.1　DOS-EXTENDER 的版本

DOS-EXTENDER 有开发版本、运行版本两种。

开发版是一个名为 RUN386. EXE 的文件，是 Phar Lap 公司的 DOS-EXTENDER 开发工具。

运行版是一个名为 RUN386B. EXE 的文件，这是用户进入实用阶段需要的版本。

### 3.1.2　DOS-EXTENDER 的相关产品

386NMM，使用 DOS-EXTENDER 的虚拟存贮系统，若希望在小内存的 PC 机上运行大的应用程序，可使用它。

### 3.1.3　DOS-EXTENDER 的适用环境

它可以运行于 80386、80486 系列的所有 PC 机之上。

### 3.1.4　DOS-EXTENDER 的兼容性

它与以下系统软件兼容：实模式和标准模式下运行的 Microsoft Windows 3. 0、Quarter-deck DESQ View 386、和所有支持 VCPI 虚拟控制程序接口的 EMS 扩展内存系统仿真程序。

Phar Lap 公司正着手新的版本以支持 OS/2、UNIX、增强模式下运行的 Windows 3. 0。

## 第二节　DOS-EXTENDER 使用简述

这里只简述其使用方法，后面 C 语言编程部分还会涉及一些具体操作。如果您不想在此花费太多时间，可跳过本节。

### 3.2.1　生成并运行保护方式的应用程序

#### 3.2.1.1　如何生成 80386 目标代码

源程序如果是用高级语言编写，则必须用可生成 80386 目标代码的编译器进行编译。在后面 C 语言编写保护方式程序部分会告诉您如何操作。

源程序如果是汇编语言，则必须用 386 汇编工具，如 phar lap assembler，386ASM，生成 80386 目标代码。

例如 汇编程序源文件 HELLO. ASM，执行如下命令：

386asm hello ↙

生成 80386 目标代码文件 HELLO. OBJ。

#### 3.2.1.2　连接目标码生成保护方式的可执行程序

对于各种方法生成的 80386 目标文件，要连接生成保护方式的可执行文件，还需要专门的连接程序，如 phar lap 的 386|LINK。

例如连接 HELLO. OBJ 和库文件 CLIB. LIB 以生成保护方式可执行文件 HELLO. EXP，操作如下：

386link hello -lib clib ↙

#### 3.2.1.3　运行保护方式的可执行程序

DOS-EXTENDER 有两种版本，其用法也各不相同。

开发版的 DOS-EXTENDER 是一个名为 RUN386. EXE 的文件，命令行语法如下：

run386　　命令行开关　　执行文件名　　命令行参数

例如

run386 hello ↙

运行版本的 DOS-EXTENDER，除一个名为 RUN386B. EXE 的文件，还有一个连接实用程序 BIND386。

用 BIND386 把您的程序和 RUN386B. EXE 连接到保护方式应用程序中，生成 . EXE 文件，能在实方式的 MS-DOS 命令行方式运行。

例如 您有一个 HELLO. EXP 文件，执行下面操作：

bind386 run386b hello ↙

则生成的 HELLO. EXE，在 MS-DOS 下运行，只需键入：

hello ↙

### 3.2.2 DOS-EXTENDER 命令行开关

利用命令行开关能改变 DOS-EXTENDER 的缺省操作。在一个命令行中,如果有若干个开关发生冲突,则最后一个有效。

DOS-EXTENDER 的所有命令行开关都在下面列出。

#### 3.2.2.1 开关名称及其语法

| 开关类别 | 全称及语法 | 缩写 |
|---|---|---|
| 空闲内存管理 | -MINREAL nparagraphs | -MINR |
| | -MAXREAL nparagraphs | -MAXR |
| DOS 缓冲 | -MINIBUF nkilobytes | -MINI |
| | -MAXIBUF nkilobytes | -MAXI |
| 物理内存使用控制 | -MAXBLKXMS nbytes | -MAXBL |
| | -MAXXMSMEM nbytes | -MAXX |
| | -MAXEXTMEM nbytes | -MAXE |
| | -MAXVCPTMEM nbytes | -MAXV |
| 扩展内存使用数量控制 | -EXTLOW address | -EXTL |
| | -EXTHIGH address | -EXTH |
| 调整特殊内存 | -NOSPCLMEM | -NOSPCL |
| | -NOBIM | -NOB |
| 线性内存使用限制 | -MAXPGMMEM nbytes | -MAXP |
| 特权级别开关 | -PRIVILEGED | -PRIV |
| | -UNPRIVIEGED | -UNPRIV |
| 混合模式程序开关 | -REALBREAK nbytes | -REALB |
| | -CALLBUFS nkilobytes | -CALLB |
| 描述符长度开关 | -GDTENT nentries | -GDTE |
| | -LDTENT nentries | -LGDE |
| 协处理器开关 | -WEITEK AUTO | -WEITEK AUTO |
| | -WEITEK ON | -WEITEK ON |
| | -WEITEK OFF | -WEITEK OFF |
| | -CYRIX AUTO | -CYRIX AUTO |
| | -CYRIX ON | -CYRIX ON |
| | -CYRIX OFF | -CYRIX OFF |
| 栈分配开关 | -NISTACK nbuffers | -NI |
| | -ISIKSIZE nkilobytes | -IS |
| 分页禁止开关 | -NOPAGE | -NOP |
| 80386 错误 17 工作区 | -ERRATE 17 | -ERR |
| PharLap 虚拟<br>内存管理程序开关 | -CODESIZE nbytes | |

| | | |
|---|---|---|
| | -DEMANDLOAD | |
| | -FLUSHSWAP | |
| | -LFU | |
| | -LOCKSTACK nbytes | |
| | -MAXSWPSIZE nbytes | |
| | -MINSWFSIZE nbytes | |
| | -NOPGEXP | |
| | -NOSWFGROWIST | |
| | -NOVM | |
| | -NUR | |
| | -PAGELOG filename | |
| | -SWAPCHK ON\|OFF\|FORCE\|MAX | |
| | -SWAPDEFDISK | |
| | -SWAPDIR dirname | |
| | -SWAPNAME filename | |
| | -SWFGROWIST | |
| | -VMFILE filename | |
| | -VSCAN nmilliseconds | |
| | -VSLEN nbytes | |
| 硬件结构类别开关 | -XT | -XT |
| | -AT | -AT |
| | -EISA | -EISA |
| 20 行地址线开关 | -A20 | -A20 |
| VDISK 兼容性开关 | -VDISK | -VDISK |
| 保留 32 位寄存器开关 | -SAVEREGS | -SAVER |
| 禁止乘法检查开关 | -NOMUL | -NOMUL |
| 在 VCPI 下不运行开关 | -NOVCPI | -NOVCPI |
| 启动调试打印开关 | -DEBUG level | -DEBUG |
| I/O 重定向开关 | -COM1 | -COM1 |
| | -COM2 | -COM2 |
| | -BAUD speed | -BAUD |
| 功能调用指针转换 | -DATATHRESHOLD nbytes | -DATAT |
| 保存实模式中断向量 | -SAVEINTS | -SAVEI |
| 重定位 BIOS 打印屏幕 | -PRIVEC vector | -PRI |
| 重定位硬中断 IRQ0～7 | -INTMAP vector | -INTM |
| | -HWIVEC vector | -HWI |
| BIOS 块传送 | -NOBMCHK | -NOBMCHK |
| 读非写模式打 | | |

| | | |
|---|---|---|
| 开 . EXP 文件 | -OPENDENY | -OPEND |

3.2.2.2 开关使用范例

| 开关名称 | 范例 |
|---|---|
| -MINREAL | run386 -minreal 100h hello |
| -MAXREAL | run386 -minr 128 -maxr 512 hello |
| -MINIBUF | run386 -minibuf 64 -maxi 64 filecopy |
| -MAXIBUF | run386 -maxibuf 2 hello |
| -MAXBLKXMS | run386 -maxbl 80000h hello |
| -MAXXMSMEM | run386 -maxx 100000h hello |
| -MAXEXTMEM | run386 -600000h hello |
| -MAXVCPTMEM | run386 -600000h hello |
| -EXTLOW | run386 -ext(on) 200000h hello |
| -EXTHIGH | run386 -ext 180000h -exth 400000h hello |
| -NOSPCLMEM | |
| -NOBIM | run386 -nobim hello |
| -MAXPGMMEM | run386 -maxp 500000h hello |
| -PRIVILEGED | run386 -priv myprog |
| -UNPRIVIEGED | run386 -upriv hello |
| -REALBREAK | run386 -realbreak 2010h -callb 2 switch |
| -CALLBUFS | run386 -realbreak 2010h -callb 2 switch |
| -GDTENT | run386 -gdte 600 -ldte 1200 hello |
| -LDTENT | run386 -gdte 600 -ldte 1200 hello |
| -WEITEK AUTO | |
| -WEITEK ON | run386 -weitek on float. exp |
| -WEITEK OFF | |
| -CYRIX AUTO | |
| -CYRIX ON | run386 -cyrix on float. exp |
| -CYRIX OFF | |
| -NISTACK | run386 -ni 8 -istk 2 switch. exp |
| -ISIKSIZE | run386 -ni 8 -istk 2 switch. exp |
| -NOPAGE | run386 -nopage numcrunch |
| -ERRATE 17 | run386 -errata numcrunch |
| -CODESIZE | |
| -DEMANDLOAD | |
| -FLUSHSWAP | |
| -LFU | |
| -LOCKSTACK | |

| | |
|---|---|
| -MAXSWPSIZE | |
| -MINSWFSIZE | |
| -NOPGEXP | |
| -NOSWFGROWIST | |
| -NOVM | |
| -NUR | |
| -PAGELOG | |
| -SWAPCHK ON\|OFF\|FORCE\|MAX | |
| -SWAPDEFDISK | |
| -SWAPDIR | |
| -SWAPNAME | |
| -SWFGROWIST | |
| -VMFILE | |
| -VSCAN | |
| -VSLEN | |
| -XT | run386 -xt hello |
| -AT | run386 -at hello |
| -EISA | run386 -eisa hello |
| -A20 | run386 -a20 hello |
| -VDISK | run386 -vdisk hello |
| -SAVEREGS | run386 -saver hello |
| -NOMUL | run386 -nomui hello |
| -NOVCPI | run386 -novcpi hello |
| -DEBUG | run386 -debug 1 hello |
| -COM1 | run386 -com1 -debug 1 hello |
| -COM2 | run386 -com2 -baud 9600 -degug 3 hello |
| -BAUD | run386 -com2 -baud 9600 -degug 3 hello |
| -DATATHRESHOLD | run386 -datat 4000h hello |
| -SAVEINTS | run386 -savei hello |
| -PRIVEC | run386 -pri FFh hello |
| -INTMAP | run386 -intm 50h hello |
| -HWIVEC | run386 -hwivec 50 hello |
| -NOBMCHK | run386 -nobmchk hello |
| -OPENDENY | run386 -opend file |

# 第四章　术语

本书用到的一些术语在此列出，供参考。

· **保护方式**

80386、80486 的 32 位工作方式。

· **DOS－EXTENDER**

它运行于 DOS 之上，提供 32 位应用程序的保护模式环境。

· **Cerious Tools**

TSR 公司提供的一个 C 语言通用工具库，能够支持 Microsoft C、Turbo C、Watcom C、Lattice C。

· **WCL386**

WATCOM 公司提供的 C 语言程序编译－链接应用软件。

· **WDISASM**

WATCOM 公司提供的目标反汇编调试应用软件。

· **命令行开关**

程序调用行上的开关，用于改变系统的缺省操作，以“-”置于开关名之前。

· **DOSX 环境变量**

硬件环境变量，DOS-EXTENDER 从它获取用户设置的命令行开关。

· **段描述符**

保护方式下各分段的定义信息都放在自己的段描述符中。

· **描述符表**

把描述符按顺序罗列在一起形成的一段数据。

· **GDT Global Descriptor Table**

全局描述符表，保护方式下的系统表，包含了整个内存空间的分段信息。

· **LDT Local Descriptor Table**

局部描述符表，保护方式下的系统表，多任务结构中，它包含了每个任务局部存贮空间的划分信息。

· **基本内存 Base Memory**

从 0 到 640KB 的内存空间。

· **扩充内存 Expanded Memory**

从 640KB 到 1MB 的内存空间。

· **扩展内存 Extended Memory**

从 1MB 开始往上的内存。

· **常规内存**

在 1MB 以内能从 MS-DOS 获得的内存。

· **直接扩展内存**

在 1MB 以上，未被分配出去的内存。

· **虚拟内存**

不一定是实际的物理内存，超出物理内存的虚拟页可以转出到磁盘上。

· **XMS 内存**

通过 XMS 接口分配的内存，XMS 驱动程序通常叫做 HIMEM. SYS。

# 第五章　在保护方式下编程

本章内容提要

. 32 位 C 程序编译

. 80386 宏汇编

. DOS-Extender

当然，保护方式下的编程工具不可能与 8086 的汇编程序以及 C 语言编译程序毫无关系，不过在 16 位编译器和 32 位编译器之间的确存在着一些区别，忽视它们，有时会导致一些灾难性的后果，比如，对于那些仅能使用基于 8086 的工具才能进行编译和连接的 C 程序。一旦您涉足 32 位 80386 保护方式编程，您会发现，编译、连接，以及运行您的 C 程序，并不比利用 16 位 8086 编程工具困难。

80386 微处理器的保护方式提供了灵活的硬件机制，以防止程序对代码段和数据段以外的访问。当一个保护方式程序在您的机器上安装并执行时，它只能对自己程序段和代码段内的数据进行存取。这种对数据进行隔离的结构在多任务环境下非常有用。例如，在多任务操作系统中，您决不希望一个程序随便去使用另一个同步运行程序的数据。

保护方式编程，对于已经习惯了 8086 方式的程序员来说，似乎有个缺陷。因为 8086 程序员习惯于硬件中含有直接访问存储器功能，而那意味着能够对程序代码段和数据段以外的内容进行读写。假如，您写了一些控制屏幕的程序，目的是把数据直接写进屏幕存储区，而且在彩色文本方式下操作，那么，数据就得写到 0X800 段。不幸的是，您如果想在保护方式下运行此程序，80386 将提示您，不要企图把数据写到代码段和数据段以外的存储空间。您将被通知程序中犯下了严重的关于存储器使用的错误，并且程序迅速被终止，由 DOS 进行处理。另一方面，如果您不得不使用 BIOS 屏幕管理子程序，那您的程序将行动迟缓且显得不那么专业。那程序员该作些什么呢？

幸运的是，有一些经常使用的存储区是与局部描述符表（LDT）硬联的，本章后面提到的 DOS-Extender 将介绍有关 LDT 的信息。Phar Lap 的 DOS-Extender 为屏幕存储建立一个 LDT 表。在第八章的屏幕控制部分您将明白如何访问用于屏幕的 LDT，以进行直接视频存储区的访问。直接视频存取意味着一个迅速有效的屏幕控制程序的出现。

这一章讨论与 C 语言相关的保护方式工具的三个基本类型：

· 32 位 C 语言编译器

· 80386 宏汇编

· DOS-Extender

在这章末尾将介绍如何设计自己的保护方式程序库。

如果现在您感兴趣的仅仅是写些平常的只使用很基本的标准库函数的 C 程序，然后用标准保护方式下的 C 编译器对其进行编译，那么您实际上不需要读这本书。然而在实际的 C 编程领域中，一个人总会慢慢对利用库生成专门有效的可执行程序产生依赖，例如用于屏幕、窗口、键盘、鼠标、图形等等操作的库函数。

如果您的库中用到汇编语言编写的函数，会怎么样呢？您可能会说，别担心！因为 80386 芯片恰好能运行所有 8086 指令。这的确减少了很多把 8086 汇编源代码转换成 80386 汇编代码的麻烦。但这并不意味着一切潜在的问题就此了结。

现在该真的讨论 32 位保护方式 C 编译了。

## 第一节　关于 32 位 80386 C 的编译

首先，我们从所有可能性出发，探讨一下 32 位编译器为何显得出类拔萃。一个较好的着眼点，看看 32 位保护方式编译器对存储管理有几种模式。

在 16 位编译器中有不同的代码段和数据段，与此对应，在 32 位 C 语言编程时有如下几种模式：

Small 和 Large 代码模式

Small 和 Large 数据模式

在 Small 代码模式中，允许使用相对于 CS 程序代码段的 32 位近调用(NEAR CALL)，就是说代码区域将被限定在 4GB 字节范围之内。为何在这么大空间内还叫作 SMALL 模式呢？其实这只不过是个相对的概念。

在 Large 代码模式中，使用 48 位远程调用(FAR CALL)，并且允许用户程序的代码部分超过 4GB 的限制。

数据模式与代码模式有相似的限制，Small 数据模式将数据限定在 4GB 范围以下，而 Small 代码模式亦被限定在 4GB 范围。

将数据段和代码段限制组合一下，存储模式共有五种可能：

| 存储模式 | 代码模式 | 数据模式 |
|---|---|---|
| Flat | Small | Small |
| Small | Small | Small |
| Medium | Large | Small |
| Compact | Small | Large |
| Large | Large | Large |

Flat 存储模式和 Small 存储模式的区别在于，Flat 模式是指代码段和数据段加在一起分配的全部存储空间必须在 4GB 范围以下，而 Small 模式是指代码段和数据段各自的空

间不超出 4GB 的限制。

本书假定您的 80386 机器上只有 2MB 字节扩展内存，所以后面的讲述均设在 Flat 存储模式下工作。请记住，4GB＝4000MB＝4000000000 个字节。

有些喜欢琢磨硬件的朋友，给自己的机器加上了 4MB 扩展内存之后，就竟然认为各种程序要运行多快就有多快。请注意，存储宽度毕竟是相对的，4MB，跟 32 位 386 保护方式下 Flat 模式的 4000MB 相比，显得多么渺小！

关于 16 位和 32 位编译器的数据表示，也有区别：

| 数据类型 | 8086 | 80386 |
| --- | --- | --- |
| char | 32 位 | 8 位 |
| short int | 16 位 | 16 位 |
| int | 16 位 | 32 位 |
| long int | 32 位 | 32 位 |

注意，在 16 位编译器下的 16 位整数表示为 int，而在 32 位编译器下，int 将代表 32 位整数。这导致的情形是，在您试图把一个 C 程序从 16 位编译器转送到 32 位编译器的过程中会发生一些古怪的问题。例如，您在程序中定义一个结构 RECT，如下：

```
typedef   struct {
          int x1;
          int y1;
          int x2;
          int y2;
          }RECT;
```

然后，您在程序中又用 RECT 说明一个叫做 R1 的变量，如下：

```
RECT R1;
```

然后，您的 C 程序库中某个地方有一汇编程序，它想接收一个指针。这个程序把一索引寄存器(SI)置成 RECT 结构的偏移值，然后取变量 X1 的值，索引寄存器(SI)再加 2，然后再取变量 y1 的值。程序如下：

```
holdx1      DW ?
holdy1      DW ?
     .
     mov     ax,[si]          ;grab R1->x1
     mov     [holdx1],ax      ;store in memory loc
     inc     si               ;add 2 to offset
     inc     si
     mov     ax,[si]          ;grab R1->y1
```

```
mov     [holdy1],ax      ;store in memory loc
 •
```

这段汇编程序在16位模式下能够很好地工作，但是它在32位模式下根本就不能正确运行。为什么？如果您在32位编译器下声明，使用32位(4字节)整数来代替16位(2字节)整数，就会变得正确。

下面，我们马上会明白有两种方法，使得汇编代码在32位编译器下恰当的工作。

第一种方法，保留原来的16位汇编代码，但要重新定义RECT结构，如下：

```
typedef   struct {
          short int x1;
          short int y1;
          short int x2;
          short int y2;
          }RECT;
```

在结构定义中把每一个变量说明成短整型(short int)，这样就能取得16位(2字节)数据，并且使汇编代码也能正常工作。

第二种方法，如果您打算改变汇编源程序，而保留C程序源代码，使RECT结构中的说明依然为32位整型。您可以按如下方式改变汇编指令，如下：

```
holdx1      DW  ?
holdy1      DW  ?
    •
    mov     ax,[esi]          ;grab R1->x1
    mov     [holdx1],ax       ;store in memory loc
    inc     esi               ;add 4 to offset
    inc     esi
    inc     esi
    inc     esi
    mov     ax,[esi]          ;grab R1->y1
    mov     [holdy1],ax       ;store in memory loc
    •
```

使用32位寄存器ESI来代替16位寄存器SI，并且给ESI寄存器加4而不是加2。加4，表示它增加4个字节，即32位。因为RECT结构中使用"int"代替" short int"来定义变量，所以每个整型变量是32位(4字节)宽。

总之，使用提供了4GB存储极限的32位386 C编译器，为保护方式下的编程提供了有力的依据。把一个8086实方式下编写的C源程序转换成运行在80386保护方式下的32位程序，我们可以思路清晰地来完成这一工作，然而在转化过程中必须细心。

# 第二节　关于 32 位 80386 汇编

这部分不准备深入讨论 80386 汇编的所有 I/O 操作，那已超出了本书重点。

下面讨论 80386 的通用寄存器以及其中一些的使用。

在 8086 芯片中，寄存器 AX 可以被写成 16 位形式 AX，或者两个 8 位寄存器的形式 AH、AL。其格式如下：

bit

| 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| |
|---|
| 15 ←——— AX ———→ 00 |

| | |
|---|---|
| 15 ←—— AH ——→ 08 | 07 ←—— AL ——→ 00 |

80386 将 16 位寄存器 AX 扩展成 32 位寄存器 EAX，这意味着用户能对 32 位的 EAX，16 位的 AX(EAX 的低位)，8 位的 AH、AL 进行操作。这是非常灵活的，例如：

inc EAX　;32 位加法是合法的

inc AX　;16 位加法是合法的

inc AH　;8 位加法是合法的

inc AL　;8 位加法是合法的

四个新的 32 位通用寄存器是：

EAX　(EAX 32 位，AX 16 位，AH 8 位，AL 8 位)

EBX　(EBX 32 位，BX 16 位，BH 8 位，BL 8 位)

ECX　(ECX 32 位，CX 16 位，CH 8 位，CL 8 位)

EDX　(EDX 32 位，DX 16 位，DH 8 位，DL 8 位)

386 汇编程序中所有标号偏移量都是 32 位宽，这与 8086 编程中 16 位偏移量相对应。这意味着，为了适应 32 位偏移量，DI，SI，BP，和 SP 四个寄存器也需要扩展，如下所示：

EDI　(EDI 32 位，DI 16 位)

ESI　(ESI 32 位，SI 16 位)

EBP　(EBP 32 位，BP 16 位)

ESP　(ESP 32 位，SP 16 位)

虽然 80386 的段寄存器仍旧保留了原来的 16 位结构，但又增加了两个新的段寄存器。80386 的所有段寄存器如下：

CS　(old)

DS　(old)

ES　(old)

FS (new)

GS (new)

SS (old)

两个新的段寄存器 FS 和 GS，不执行任何特殊功能，它们的用法与 ES 一样。

最后，还有寄存器 IP 和 PLAGS 也已经被扩展，如下：

EIP (EIP 32 位，IP 16 位)

EFLAGS (EFLAGS 32 位，FLAGS 16 位)

80386 不仅支持所有旧的 8086 寻址模式，而且又增加了新的寻址模式。考虑到 80386 汇编，本书集中精力重新整理了 8086 汇编例行子程序，以便 80386 的 32 位方式程序能够调用它们。更准确地说，我们将致力于把现有的 8086 汇编子程序转换成 80386 汇编子程序。

尽管后面将给出把子程序从 8086 转化到 80386 的精巧别致的方法，但这里还有必要强调两点。

第一点涉及 80386 如何控制循环。我们先看一个 8086 汇编代码片断：

```
        •
        mov CX,10
rep     movsb
        •
```

这里，寄存器 CX(计数器)被置初值，然后“movsb”指令重复执行 10 遍。把这段程序拿到 80386 方式下执行，您认为能行吗？别急于回答，我们先看看这段代码在 80386 方式下的写法：

```
        •
        mov ECX,10
rep     movsb
        •
```

80386 的循环机制将 ECX 看作计数寄存器，如果您仅仅在 CX 中装入计数值，那么 ECX 的高字位置的值将保留不变。假如 ECX 的高字位置的值不等于 0FFH，那么您的汇编程序显然会出现灾难性结局。进行 80386 汇编程序设计时，若遇到循环或者重复执行的指令序列，千万别粗心大意。牢记计数寄存器是 ECX 不是 CX，就不会出错了。

另一点，提醒您在使用 BIOS 或 DOS 时注意。例如下面这一个 16 位汇编片断：

```
          •
hello DB ' Hello Chuck! ' ,0ah,0dh,' $'
          •
    lea DX,hello        ;load offset hello to DX
    mov AH,9            ;and use DOS to print string
    int 21h
          •
```

在 80386 汇编方式下，应该改写成：

```
    .
hello DB ' Hello Chuck! ' ,0ah,0dh,' $'
    .
  lea EDX,hello      ;load offset hello to DX
  mov AH,9           ;and use DOS to print string
  int 21h
    .
```

总之，记清，在循环或重复执行的指令序列中，要设置ECX，而不只是CX。再者，记清，80386 汇编标号的偏移量都是 32 位，并且，要把偏移量装载到扩展的寄存器(如 EDX)而不是 16 位的寄存器(如 DX)上。

## 第三节　DOS-EXTENDER

DOS 从未被设计成一个保护方式操作系统。

DOS-Extender 扩展了一般的 DOS 功能，使之能在 80386 保护方式环境中工作。您将会发现，使用 DOS-Extender，很容易把 C 程序从实方式 DOS 环境转向保护方式，这恰好对应于把您的程序转向一个新的通用操作系统，如 OS/2。

对一个编写商业性程序的人来讲，关于"相对于新一代操作系统的 DOS-Extender"的出现，还有一点需要考虑，实际上所有 80386 机器都有对 DOS 的访问，而当前极少有机器具备新一代保护方式操作系统。也就是说作为一个商业性的程序员，不单要想到如何写程序，还要思考用户怎样来运行程序。DOS-Extender 简化了保护方式程序在配备 1MB 以上扩展内存的标准 386 机器上的运行。

实质上，DOS-Extender 截取了 BIOS 或 DOS 系统调用，并且通过 80386 的实方式来发送。重复从保护方式弹回到实方式，也许会降低程序性能。如果有经验的话，您也许也有一种感觉，386 保护方式似乎并不比 8086 实方式来得敏捷，那就是说，一个 32 位指令的 80386 程序在保护方式下执行，会比同样功能的 16 位指令程序在 8086 实方式下运行得"慢"，因此，重要的是，当您使用 DOS-Extender 调用 BIOS 或 DOS 时要提起注意，因为它们会降低程序执行速度。

有两个基本的方法，能够应用 DOS-Extender 使您的程序运行方便易行。

第一个方法，从命令行执行 DOS-Extender。本书中所有程序例子的运行，都是使用 Phar Lap 的命令行版本 DOS-Extender。在下一章您将学会编写两种版本的您现在还不熟悉的"Hello Chuck!"程序。两种版本即汇编语言和 C 语言，它们都需要 DOS-Extender 去运行。

C 版本的"Hello Chuck!"程序被编译、连接之后，会生成一个文件 Hello. EXP。文件扩展名. EXP 为保护方式程序所有。使用 Phar Lap 的 DOS-Extender，很好运行 Hello. EXP，在命令行键入下列命令，然后按 Enter 键，如下：

RUN386　HELLO

执行保护方式程序的这第一个方法，要求您必须具备一份自己的 DOS-Extender 拷

贝。

第二个方法，就个人喜好而言，这也是个比较不错的方法，即利用 DOS-Extender 的连接程序。它把您的.EXP 文件与 DOS-Extender 内核连接在一起，并且产生一个标准的 DOS 版 .EXE 文件。当这被连接好的.EXE 文件执行时，它把 80386 切换到保护方式。

DOS-Extender 扩展程序作为自己的实用程序和 DOS 之间的接口，容许您访问绝大部分 BIOS 与 DOS 系统调用。目前还没有一个标准告诉 DOS-Extender 的编制者，哪些 BIOS 与 DOS 系统调用必须被支持，哪些可能需要支持。

本章稍前一点曾提到过，DOS-Extender 允许利用 LDT 局部描述符表访问机器中任何存储空间。如果为进行直接视频存储访问而使用 Phar Lap 的 LDT 编写保护方式程序，它将不能在 A · I · Architect 的 DOS-Extender 下很好地运行，为什么呢？因为 A · I · Architect 的 DOS-Extender 用于屏幕显示的 LDT 段空间与 Phar Lap 不同。如果您希望自己的程序既能在 Phar Lap 又能在 A · I · Architect 的 DOS-Extender 下运行，您可经常利用屏幕 BIOS 读写，但是程序性能会因此受损。

其实，最好选定一种且只能一种 DOS-Extender，来运行自己的程序，本书的全部示范程序就都使用了 Phar Lap 的 DOS-Extender。

总之，DOS-Extender 把 DOS 装扮成了一个极其有效的操作系统。利用 DOS-Extender 能够获得大部分 BIOS 和 DOS 系统调用。唯一不幸的是，并非所有类型的 DOS-Extender 都彼此等价，这意味着您的保护方式程序也许不能在所有类型 DOS-Extender 的支持下运行。

## 第四节　设计保护方式程序库

建立自己的库函数，其意义是不言而喻的，尤其在编写稍大一点的 C 程序时，这一点将显得更加突出。开发 C 程序库，首先得考虑的事情就是，使库程序的函数调用在源代码级上和实方式的库程序兼容。这就使得 16 位 8086 C 语言程序到 32 位 80386C 语言程序的移植变得容易。

一旦决定了库函数的类型，下面的事情就是选择工具了。Watcom C 就挺受人欢迎。令人兴奋的是 Watcom C 推出了 32 位指令保护方式程序的编译器，而且 Novell 选择了 Watcom C 386 作为他们的网络编译器。Watcom 386 C 8.0 在国外的用户中拥有良好的印象，而且如今亦称得起一个性能卓越的产品。本书中描述并使用了一个特别强大的编译和连接程序软件包的比较实在的部分，这个软件包含了大量扩展的应用程序。

Watcom 在保护方式 C 编译器与 Microsoft 兼容方面所作的努力，最终使得库函数从 8086 实方式到保护方式的转换异常轻松简单。

尽管 Phar Lap 的宏汇编和连接程序都是优质高级的工具，但 Watcom C 正准备提供一个 8.0 版的连接程序，而它与 Microsoft 的兼容性，正是 Microsoft MASM 用户求之不得的。

对于 DOS-Extender，在 Phar Lap 和 A. I. Architect 之间，我们选择了前者。

概要：

总结一下，至此我们已做了些什么事情

1. 确定自己的库的内容

2. 挑出一个 C 编译器

3. 选择一个宏汇编程序

4. 认准一个 DOS-Extender

至此，为了开始实际的编程，已经介绍了足够的背景资料。在下章，我们就将编写并运行保护方式程序。

# 第六章　编写并运行保护方式程序

内容提要

- 用 C 语言编写保护方式程序
- C 程序调用 80386 汇编子程序

在这一章，您将编写并运行第一个保护方式程序。例程全部采用 Watcom 386 C v8. 0。正如曾经提到过的，Watcom 编译器超群的强大和灵活。本书着眼于为保护方式编译器建造 C 程序库，并且介绍一些与本文有关的编译器的特点。

象所有新一代编译器一样，Watcom 提供了一个“编译—连接”应用程序，WCL386 。它允许您使用恰当的命令调用编译器，然后把所有编译好的目标模块连接在一起。很多手册都给出了此编译—连接应用程序，然而在这一章，有关如何变动一些编译选项以及如何调用功能函数的介绍依然显得特别重要。

本章第二节将提供一些从目标反汇编得来的信息，以说明汇编子程序对参数传送的约定。

## 第一节　用 C 语言编写保护方式程序

安装好编译器，然后就可以编译并连接您的第一个程序了。本书使用了 Watcom 386C v8. 0 版的编译器。

第一个程序 HELLO1. C 见程序 6—1. 其功能很简单，即在屏幕上打印出字符串

Hello Chuck!

然后返回 DOS。程序清单见程序 6—1.

程序 6—1. HELLO1. C 程序源代码清单

```
//////////////////////////////////////////////////
// HELLO1. C                                    //
// Description:                                 //
//          Prints "Hello Chuck!" to the screen //
//////////////////////////////////////////////////
//include files here
#include <stdio. h>

//global data
```

```
char hello[]={"Hello Chuck!"};

//function declarations
void main(void);
void show-mess(char * );

void show-mess(char * mess)
{
      puts(mess);
}

void main()
{
      show-mess(hello);
}
//////////////////////////////////////////////////////
```

HELLO1.C 是一个很简单的程序，包括两个函数。程序执行从函数 main() 开始，而函数 show-mess(...)用于打印字符型指针传送过来的字符串。

现在编译、连接此保护方式程序，键入

WCL386 HELLO1.C /3s

然后按 Enter 键。开关 /3S 告诉编译器使用堆栈传递参数。象 Microsoft C、Turbo C、以及当今市场上其它众多编译器，用堆栈传递参数都是标准的。

用 WCL386 编译且连接好 HELLO1.C 之后，生成一个名为 HELLO1.EXP 的文件，文件长度 6,748 个字节。用 Phar Lap 的 DOS－Extender 程序来运行 HELLO1.EXP，键入

RUN386 HELLO1

然后按 Enter 键。如果运行正常，屏幕上会显示如下信息

Hello Chuck!

Watcom 提供给程序员的调试工具是 WDISASM，一个目标模块反汇编程序。WDISASM 对于我们了解编译器如何把 C 源程序变成汇编代码，价值极大。

我们反汇编一下 HELLO1.OBJ，键入

WDISASM  HELLO1.OBJ ＞ HELLO1.LST

按 Enter 键。WDISASM 将把一个目标列表反汇编结果写入文件 HELLO1.LST。程序 6－2. 给出了 HELLO1.OBJ 使用堆栈传递参数情形时的目标反汇编程序。

程序 6—2. 使用堆栈传递参数情形时 HELLO1. OBJ 的目标反汇编清单

```
Module: hello1.c

Group: 'DGROUP'  CONST,-DATA,-BSS

Segment: '-TEXT'  BYTE USE32 00000036 bytes
0000 50                       show-mess   push eax
0001 b8 0c 00 00 00                       mov eax,0000000ch
0006 e8 00 00 00 00                       call -STK
000b 58                                   pop eax
000c 55                                   push ebp
000d 89 e5                                mov ebp,esp
000f ff 75 08                             push +8h[ebp]
0012 e8 00 00 00 00                       call puts
0017 83 c4 04                             add esp,0004h
001a c9                                   leave
001b c3                                   ret
001c 50                                   push eax
001d b8 08 00 00 00                       mov eax,00000008h
0022 e8 00 00 00 00                       call -STK
0027 58                                   pop eax
0028 68 00 00 00 00                       push offset hello
002d e8 ce ff ff ff                       call show-mess
0032 83 c4 04                             add esp,0004h
0035 c3                                   ret

No disassembly errors

Segment: 'CONST'  WORD USE32 00000004 bytes
0000 00 00 00 00    -main-entry-   DD    -NULL-Argv

No disassembly errors

Segment: '-DATA'  WORD USE32 0000000d bytes
0000   48 65 6c 6c 6f 20 43 68 hello     - Hello Ch
0008   75 63 6b 21 00                    - uck!.
```

No disassembly errors

检查一下程序 6—2. 中的目标反汇编列表程序，您会发现一些有趣的事情。在调用函数 show-mess 的汇编片断中加进如下内容

```
.
push offset hello
call show-mess
add esp, 0004h
.
```

函数 show-mess(...) 取出一个参数，它是一字符指针(CHAR * )。hello 的 32 位偏移量被压进 ESP 寄存器，而且函数 show-mess(...) 被调用之后，ESP 寄存器加 4。

在函数 show-mess(...)中放进如下一段指令

```
.
push esp              ;set up stack frame
mov ebp, esp
push +8[ebp]          ;push 32 bit char * parameter on stack
call  puts            ;C library function
add  esp,0004h        ;restore esp value
.
```

指向 hello 的字符指针(CHAR * )从[EBP+8] 取出。也就是说第一个参数的偏移量是 ESP+8 ，而且压进栈中的是 32 位数。

重新编译、连接程序 HELLO1. C，这次使用寄存器参数传递方式的编译选项，键入如下 WCL386 命令行：

WCL386 HELLO1. C /3r ↙

注意，开关/3r ，告诉编译器使用寄存器传递参数，它在 WCL386 编译—连接应用程序中是缺省设置的。

这次用 WCL386 /3r 编译且连接完 HELLO1. C，您会发现 HELLO1. EXP 文件长度是 6,300 个字节，这比用 WCL386 /3s 得出的 HELLO1. EXP 少 448 个字节。程序 6—3. 给出了使用寄存器传递参数的 HELLO1. OBJ 反汇编列表文件清单。

程序 6—3. 使用寄存器传递参数情形时 HELLO1. OBJ 的目标反汇编清单

```
Module: hello1.c

Group: ' DGROUP'  CONST,-DATA,-BSS

Segment: ' -TEXT'  BYTE USE32 00000024 bytes
0000   50                show-mess  push   eax
0001   b8 04 00 00 00               mov    ax,00000004h
```

```
0006    e8 00 00 00 00                call   -STK
000b    58                            pop    eax
000c    e9 00 00 00 00                jmp    puts-
0011    50                   main-    push   eax
0012    b8 04 00 00 00                mov    eax,00000004h
0017    e8 00 00 00 00                call   -STK
001c    58                            pop    eax
001d    b8 00 00 00 00                mov    eax,offset-hello
0022    eb dc                         jmp    show-mess-

No disassembly errors

Segment: ' CONST'  WORD USE32 00000004 bytes
0000     00 00 00 00  -main-entry-  DD  -NULL-Argv

No disassembly errors

Segment: ' -DATA'  WORD USE32 0000000d bytes
0000    48 65 6c 6c 6f 20 43 68 hello      -Hello Ch
0008    75 63 6b 21 00                     -uck!.

No disassembly errors
```

反汇编程序中指明了，hello 的 32 位偏移量放在 32 位寄存器 EAX 中。用寄存器传递参数可靠而且迅速。寄存器参数传递让人感觉舒适的是函数长度和性能。而当您关心的是汇编源代码的维护时，则倾向于堆栈参数传递。

程序员建库时，经常会遇到一种“两难”境界。即我们面临两个选择，两者都有可取之处，而两个选择也都有令您讨厌的地方，选中其一，就会失去另一个的优点。这在汇编子程序上有着突出的表现。

Cerious Tools V2.0(TSR 公司的支持 Microsoft C、Turbo C、Watcom C、和 Lattice C 的 C 语言通用函数库)包括了约 135 个汇编模块和 135 个 C 模块。现在，若突然出现一个故障或在 C 模块中出现一个体系问题，那只好用合适的编译器重新编译并且重新放到所有合适的库中。

而当一个汇编模块被修改时，情况就不同了。这是由于，如果编程时使用了 Microsoft C 快速寄存器调用约定、Watcom C 缺省的寄存器调用约定以及用于 Turbo C 或 Lattice C 的堆栈调用约定，那我们将写出三种不同的参数传递方案。这就跟所有编译器都支持堆栈参数传递一样，只不过是使用一套汇编模块来适应所有编译器。

虽然使用堆栈参数传递从源代码维护角度来看觉得不错，可是它会使源代码模块庞

大、缓慢。假如期望拥有长度又小性能又佳的源代码，那么源代码维护又成了一个令人头疼的麻烦。真是进退两难！而且目前多数编译器都在很大程度上与 Microsoft C 的堆栈参数传递约定相兼容。

似乎妥协一下，为汇编子程序解决解决参数传递策略，就能缓解这两难境界。本章下一节，您将学会如何用参数传递约定使每个 C 库函数快速、有效，以及堆栈参数传递如何使得汇编函数源代码的维护变得容易。

## 第二节　C 程序调用 80386 汇编子程序

在介绍使用 80386 汇编语言编制 C 程序的函数之前，先提供一个用 C 语言编写的很简单的函数，然后再用汇编语言重写此函数。C 语言函数将作为一个基本的界线，告诉我们是否值得费劲去把它转换成汇编语言模块。

函数 mvCur(...) 出现在程序 MVCUR.C(见程序 6—4.)中，它按如下方式进行操作：

1. 使用 BIOS 的 int 10h 第 0Fh 号功能调用，把当前显示页面放到寄存器 BH 中。

2. 使用 BIOS 的 int 10h 第 02h 号功能调用，移动光标。

注意，16 位 C 编译器使用界面调用函数 int86(...) 调用 BIOS 。Watcom C v8.0 用的是 int386(...) ，尽管从源代码兼容性角度来说，我们希望保持 int86(...) 这个名称，但显而易见 Watcom 使用名称 int386(...) 是有用意的。

使用 WCL386 编译 MVCUR.C ，键入

```
WCL386 MVCUR.C /c/3s
```

开关/c 告诉 WCL386 只编译 C 函数。连接作用在此被忽略。开关 /3s 告诉编译器使用堆栈传递参数。(这并非编译 C 函数的最有效的方法)。

程序 HELLO2.C (见程序 6—5.)，展示了经过改动的 show-mess(...) 函数，改动后用来接收 3 个参数。前两个参数是光标的行号和列号，第三个参数是一个指针，它指向准备显示的字符串。

使用 WCL386 编译-连接应用程序，很容易建立多目标模块的程序。在命令行键入

```
WCL386 HELLO2.C MVCUR.OBJ /3s
```

按 Enter 键，HELLO2.C 就会被编译，并且与 MVCUR.OBJ 连接形成一个叫做 HELLO2.EXP 的程序。HELLO2.EXP 是用纯粹堆栈参数传递方法编译和连接的。看看 HELLO2.OBJ 的目标反汇编程序(见程序 6—6.)，就会弄清楚是如何用栈进行实际的参数传递的。

注意函数 show-mess(...) 是如何控制参数的。参数由栈中取出，然后当 mvCur(...) 和 puts(...) 被调用时，参数被压入栈中。

MVCUR.OBJ 的目标反汇编程序在程序 6—7. 中给出。注意在函数 int386(...) 被调用前，函数 mvCur(...) 如何从栈中取参数，又是如何将参数压进堆栈的。

程序 6—4. MVCUR.C 源代码清单

```
//////////////////////////////////////////
// MVCUR.C                              //
// Description:                         //
// Move the cursor via BIOS             //
// Entry:                               //
// short int row => row location        //
// short int col => column location     //
// Exit:                                //
// Returns nothing                      //
//////////////////////////////////////////
// include files
#include <dos.h>

//function prototype
void mvCur(short int ,short int);

//function
void mvCur(short int row,short int col)
{
      union REGS ir,or;

      //get current video page to BH register
      ir.h.ah = 0x0f;
      int386(0x10,&ir,&or);

      // move the cursor
      ir.h.ah = 0x02;
      ir.h.dl = (char)col;
      ir.h.dh = (char)row;
      int386(0x10,&ir,&or);
}
//////////////////////////////////////////
```

HELLO2.EXP 长度 7,948 个字节，在前面介绍 386 C 保护方式程序时，就说过使用哪一种参数传递方式能产生较短的执行代码，当然是寄存器传递，而不是堆栈传递。MVCUR.C 被稍微改动后的情形见程序 6—8. 从中可看出发生了两件事情：

1. STDDEF.H 文件被包含进去了

2. mvCur 的函数说明由 void mvCur(short int,short int) 改成了

   void cdecl mvCur(short int,short int)

程序 6—5. HELLO2.C 程序源代码清单

```
//////////////////////////////////////////////
// HELLO2.C                                 //
// Description:                             //
// Move the cursor                          //
// Prints "Hello Chuck!" to the screen      //
//////////////////////////////////////////////
//include files here
#include <stdio.h>

//global data
char hello[]={"Hello Chuck!"};

//function declarations
void mvCur(short int,short int);
void main(void);
void show-mess(short int,short int ,char * );

// move the cursor to specfied location and
// print a string

void show-mess(short int row,short int col,char * mess)
{
     mvCur(row,col);
     puts(mess);
}

void main()
{
     // Move the cursor to row 2,column 34
     // and print the "Hello" message
     show-mess(2,34,hello);
}
//////////////////////////////////////////////
```

程序 6－6. HELLO2. OBJ 目标反汇编程序

```
Module: hello2. c

Group ' DGROUP'   CONST,-DATA,-BSS

Segment: ' -TEXT'   BYTE USE32 00000050 bytes
0000   50                 show-mess   push eax
0001   b8 14 00 00 00                 mov eax,00000014h
0006   e8 00 00 00 00                 call -STK
000b   58                             pop eax
000c   55                             push ebp
000d   89 e5                          mov ebp,esp
000f   66 8b 55 08                    mov dx,+8h[ebp]
0013   66 8b 45 0c                    mov ax,+och[ebp]
0017   98                             cwde
0018   50                             push eax
0019   0f bf c2                       movsx eax,dx
001c   50                             push eax
001d   e8 00 00 00 00                 call mvCur
0022   83 c4 08                       add esp,0008h
0025   ff 75 10                       push +10h[ebp]
0028   e8 00 00 00 00                 call puts
002d   83 c4 04                       add esp,0004h
0030   c9                             leave
0031   c3                             ret
0032   50                 main        push eax
0033   b8 10 00 00 00                 mov eax,00000010h
0038   e8 00 00 00 00                 call -STK
003d   58                             pop eax
003e   68 00 00 00 00                 push offset hello
0043   6a 22                          push 22h
0045   6a 02                          push 02h
0047   e8 b4 ff ff ff                 call show-mess
004c   83 c4 04                       add esp,000ch
004f   c3                             ret

No disassembly errors
```

```
Segment:' CONST'  WORD USE32 00000004 bytes
0000  00 00 00 00  -main-entry-   DD   -Null-Argv
```

No disassembly errors

```
Segment: ' -DATA'  WORD USE32 0000000D bytes
0000  48 65 6c 6f 20 43 48 hello     -HEllo Ch
0008  75 63 6b 21 00                 - uck!.
```

No disassembly errors

程序 6—7. MVCUR.OBJ 目标反汇编程序清单

```
Module: mvcur.c

Group ' DGROUP'  CONST,-DATA,-BSS

Segment: ' -TEXT'  BYTE USE32 00000054 bytes
0000  50                  -mvCur  push eax
0001  b8 54 00 00 00              mov eax,00000054h
0006  e8 00 00 00 00              call -STK
000b  58                          pop eax
000c  0f a0                       push fs
000e  0f a8                       push gs
0010  55                          push ebp
0011  89 e5                       mov ebp,esp
0013  83 ec 38                    sub esp,0038h
0016  c6 45 c9 0f                 mov byte ptr -37h[ebp],0fh
001a  8d 45 e4                    lea eax,--1ch[ebp]
001d  50                          push eax
001e  8d 45 c8                    lea eax,-38h[ebp]
0021  50                          push eax
0022  6a 10                       push 10h
0024  e8 00 00 00 00              call int386
0029  83 c4 0c                    add esp,000ch
002c  c6 45 c9 02                 mov byte ptr -37h[ebp],02h
0030  8a 45 14                    mov al,+14h[ebp]
0033  88 45 d4                    mov -2ch[ebp],al
```

```
0036   8a 45 10             mov al,+10[ebp]
0039   88 45 d5             mov -2bh[ebp],al
003c   8d 45 e4             lea eax,-1ch[ebp]
003f   50                   push eax
0040   8d 45 c8             lea eax,-38h[ebp]
0043   50                   push eax
0044   6a 10                push 10h
0046   e8 00 00 00 00       call int386
004b   83 c4 0c             add esp,000ch
004e   c9                   leave
004f   0f a9                pop gs
0051   0f a1                pop fs
0053   c3                   ret

No disassembly errors
```

头文件 STDDEF. H 定义了 cdecl。当 cdecl 被放到函数 prototype 中时,它告诉 Watcom C v8.0 用堆栈传递参数方法编译此函数。后面给出 HELLO3. C 时,这些都已经完成。程序 6-8. 提供了更新过的 MVCUR 程序清单。

程序 6-8. MVCUR. C 更新后的源代码清单

```
////////////////////////////////////////////
// MVCUR.C                                //
// Description:                           //
// Move the cursor via BIOS               //
// Entry:                                 //
// short int row => row location          //
// short int col => column location       //
// Exit:                                  //
// Returns nothing                        //
////////////////////////////////////////////
// include files
#include <dos.h>
#include <stddef.h>

//function prototype
void mvCur(short int ,short int);
```

```
//function
void cdecl mvCur(short int row,short int col)
{
      union REGS ir,or;

      //get current video page to BH register
      ir.h.ah = 0x0f;
      int386(0x10,&ir,&or);

      // move the cursor
      ir.h.ah = 0x02;
      ir.h.dl = (char)col;
      ir.h.dh = (char)row;
      int386(0x10,&ir,&or);
}
//////////////////////////////////////////
```

HELLO3.C 的源程序清单见程序 6—9.现在对 mvCur(...)的函数说明，在函数名前加上了 cdecl。用 cdecl 说明 mvCur(...)，就可以用寄存器参数传递方式来编译 HELLO3.C 了。然而当函数 mvCur(...)被调用时，参数是通过堆栈传送的。简言之，Watcom C v8.0 允许您自己给每个函数方便地定义参数传递方式。如果您决定仅使用 Watcom C v8.0 编译器的话，那您就可以利用汇编语言写出寄存器参数传递方式的高度优化的程序。如果不是这样的话，请最好坚持如下两点作法：

- 所有 C 函数接收参数，使用寄存器
- 所有汇编函数接收参数，使用堆栈

程序 6—9. HELLO3.C 程序源代码清单

```
////////////////////////////////////////////////
// HELLO3.C                                  //
// Description:                              //
// Move the cursor                           //
// Prints "Hello Chuck!" to the screen       //
////////////////////////////////////////////////
//include files here
#include <stdio.h>
#include <stddef.h>

//global data
```

```
char hello[]={"Hello Chuck!"};

//function declarations
void cdecl mvCur(short int,short int);
void main(void);
void show-mess(short int, short int, char * );

// move the cursor to specfied location and
// print a string
void show-mess(short int row,short int col,char * mess)
{
     mvCur(row,col);
     puts(mess);
}

void main()
{
     // Move the cursor to row 2,column 34
     // and print the "Hello" message
     show-mess(2,34,hello);
}
//////////////////////////////////
```

编译 HELLO3.C 并且与 MVCUR.OBJ 连接,用下述命令:

WCL386 HELLO3.C MVCUR.OBJ /3r ↙

WCL386 将编译、连接在 HELLO3.EXP 中所有使用寄存器参数传递的函数,不过 mvCur(...)例外。程序 6—10. 给出了 HELLO3.OBJ 的目标反汇编程序。注意 函数 show-mess(...) 如何从寄存器中接收参数,如何把参数放到栈上供函数 mvCur(...) 使用,以及当 puts(...)被调用时如何把一个参数放到寄存器中。

程序 6—10. HELLO3.OBJ 目标反汇编程序

```
Module: hello3.c

Group       'DGROUP'  CONST,-DATA,-BSS

segment:       '-TEXT'  BYTE USE32 00000052 bytes
0000  50                     show-mess-   push eax
0001  b8 14 00 00 00                      mov eax,00000014h
```

```
0006  e8 00 00 00 00              call -STK
000b  58                          pop eax
000c  51                          push ecx
000d  56                          push esi
000e  66 89 c1                    mov cx,ax
0011  89 de                       mov esi,ebx
0013  0f bf c2                    movsx eax,dx
0016  50                          push eax
0017  0f bf c1                    movsx eax,cx
001a  50                          push eax
001b  e8 00 00 00 00              call -mvCur
0020  83 c4 08                    add esp,0008h
0023  89 f0                       mov eax,esi
0025  e8 00 00 00 00              call puts-
002a  5e                          pop esi
002b  59                          pop ecx
002c  c3                          ret
002d  50                 main-    push eax
002e  b8 0c 00 00 00              mov eax,0000000ch
0033  e8 00 00 00 00              call -STK
0038  58                          pop eax
0039  53                          push ebx
003a  52                          push edx
003b  b8 02 00 00 00              mov eax,00000002h
0040  ba 22 00 00 00              mov edx,00000022h
0045  bb 00 00 00 00              mov ebx,offset-hello
004a  e8 b1 ff ff ff              call show-mess-
004f  5a                          pop edx
0050  5b                          pop ebx
0051  c3                          ret

No disassembly errors

Segment:' CONST'  WORD USE32 00000004 bytes
0000  00 00 00 00     -main-entry-  DD  -Null-Argv-

No disassembly errors
```

```
Segment: ' -DATA'  WORD USE32 0000000d bytes
0000   48 65 6c 6f 20 43 48 -hello        -hello Ch
0008   75 63 6b 21 00                     -uck!.
```

No disassembly errors

MVCUR.OBJ 的目标反汇编在程序 6—11. 中给出。注意，在函数 int386(...)被调用之前，函数 mvCur(...)从栈上取数，而返回参数放到寄存器中。

程序 6—11. MVCUR.OBJ 目标反汇编程序

```
Module: mvcur.c

Group ' DGROUP'  CONST,-DATA,-BSS

Segment: ' -TEXT'  BYTE USE32 00000048 bytes
0000  50                 -mvCur  push eax
0001  b8 48 00 00 00             mov eax,00000048h
0006  e8 00 00 00 00             call -STK
000b  58                         pop eax
000c  55                         push ebp
000d  89 e5                      mov ebp,esp
000f  83 ec 38                   sub esp,0038h
0012  c6 45 c9 0f                mov byte ptr -37h[ebp],0fh
0016  b8 10 00 00 00             mov eax,00000010h
001b  8d 55 c8                   lea eax,-38h[ebp]
001e  8d 5d e4                   lea ebx,-1ch[ebp]
0021  e8 00 00 00 00             call int386-
0026  c6 45 c9 02                mov byte ptr -37h[ebp],02h
002a  8a 55 0c                   mov dl,+0ch[ebp]
002d  88 55 d4                   mov -2ch[ebp],dl
0030  88 55 08                   mov dl,+8h[ebp]
0033  88 55 d5                   mov -2bh[ebp],dl
0036  b8 10 00 00 00             mov eax,00000010h
003b  8d 55 c8                   lea edx,-38h[ebp]
003e  8d 5d e4                   lea edx,-1ch[ebp]
0041  e8 00 00 00 00             call int386-
0046  c9                         leave
0047  c3                         ret
```

No disassembly errors

在给出汇编版本的函数 mvCur(...)之前，先比较一下 HELLO2. EXP 和 HELLO3. EXP 两个文件的大小

| 程序名称 | 文件大小 | 参数传递方式 |
|---|---|---|
| HELLO2. EXP | 7,948 字节 | 堆栈 |
| HELLO3. EXP | 7,500 字节 | 寄存器 * |

* 函数 mvCur(...)从堆栈中取用参数

您可以清楚地看出，寄存器参数传递方式总是节省 448 个字节。

我们还可以进一步缩短程序长度，而总是使其性能有所增强。方法是，把用 C 语言编写的函数转化成汇编语言函数。当然，汇编语言程序总是比 C 语言程序难以维护，不过一旦您手上拥有一个好工具软件的话，维护也似乎不是什么为难的事情。

开始编写 80386 汇编子程序的最简单的方法，就是把一个简单的 BIOS 程序从 C 转移到汇编。前面给出的函数 mvCur(...)，是个可以着手的好地方。

汇编文件 MVCUR. ASM，见程序 6—12. 将使用 Phar Lap 的宏汇编 386ASM 来汇编。MVCUR. ASM 用的是一个较老类型的汇编格式。很简单，-mvCur 开始先用 EBP 和 ESP 两个 32 位寄存器设置堆栈结构，这对应于 8086 汇编中的寄存器 BP 和 SP。而在程序内部，汇编代码与 8086 代码是一致的。第一个参数的偏移量是 8。看下述格式：

参数 1 → [EBP+08]

参数 2 → [EBP+12]

参数 3 → [EBP+16]

如果想用 Phar Lap 的 386ASM 汇编 MVCUR. ASM，键入

386ASM MVCUR

按 Enter 键，然后迅速产生一个目标文件 MVCUR. OBJ。

文件 MVCUR. ASM 更新后的源代码清单(见程序 6—13.)，可以用 MASM 5. 0+ 版本进行汇编。它使用了一些新型指令。注意，在 MASM 5. 1 中，USES 指令、或者在 PROC 之后传递的参数都不给压入 EBP 寄存器的参数产生 32 位偏移量。

您会看到，-mvCur 过程更新后的基本结构与原来的 MVCUR. ASM 是一致的。更新形式的 MVCUR. ASM 使用了新型的 DOS 段(DOSSEG)、模式(MODEL)、以及代码(CODE)指令来说明段和模式。请注意，文件头上的指令 ·386，它告诉 MASM 接收 80386 指令。

程序 6－12. MVCUR. ASM 源代码清单

```
;///////////////////////////////////////
;// MVCUR. ASM                       //
;// Description:                     //
;// Move cursor                      //
;// Entry:                           //
;// P1 short int row                 //
;// P2 short int col                 //
;///////////////////////////////////////
;parameters
row EQU [EBP+8]
column EQU [EBP+12]

;declare text segment
-text segment byte public ' code'
assume cs:-text

;declare -mvCur public

          PUBLIC -mvCur

;begin procedure
-mvCur   proc
          push EBP                  ;set stack frame
          mov EBP,ESP
          mov AH,0FH                ;get current vid page to BH
          int 10h                   ;via BIOS
          mov dh,row                ;parameter 1 ->DH
          mov dl,column             ;parameter 2 ->DL
          mov ah,2                  ;move cursor
          int 10h                   ;via BIOS
          mov ESP,EBP               ;restore
          pop EBP
          ret
-mvCur    endp
-text   ends
          end
;//////////////////////////////////////////////////
```

对更新后的 MVCUR. ASM 进行汇编，键入

MASM /ML MVCUR ;

按 Enter 键。开关/ML 告诉 MASM 将 PUBLIC 置于敏感状态。恰好 C 语言也对此敏感。然而，有些连接程序的缺省值将此敏感状态开关关闭。在准备汇编语言函数时，打开此开关并不是个坏习惯。注意安全总不是坏事，打开敏感开关总不会受害，但是，假如您的连接程序缺省值表示打开，那关闭此开关也许会带来麻烦。

程序 6—13. MVCUR. ASM 更新后的源代码清单

```
;//////////////////////////////////////////////
;// MVCUR. ASM                              //
;// Description:                            //
;// Move cursor                             //
;// Entry:                                  //
;// P1 short int row                        //
;// P2 short int col                        //
;//////////////////////////////////////////////
;declare 80386 code
.386

;parameters
row EQU [EBP+8]
column EQU [EBP+12]

;declare mode & code segment
DOSSEG

.MODEL SMALL

.CODE

;declare -mvCur public

            PUBLIC -mvCur

;begin procedure
-mvCur   proc
```

```
          push EBP          ;set stack frame
          mov EBP,ESP
          mov AH,0FH        ;get current vid page to BH
          int 10h           ;via BIOS
          mov dh,row        ;parameter 1 —>DH
          mov dl,column     ;parameter 2 —>DL
          mov ah,2          ;move cursor
          int 10h           ;via BIOS
          mov ESP,EBP       ;restore
          pop EBP
          ret
-mvCur    endp
          end
;///////////////////////////////////////////////
```

程序 6－14. 给出了用 Phar Lap 产生的目标文件 MVCUR. OBJ 的目标反汇编程序清单。程序 6－15. 给出了由 MASM 产生的目标文件 MVCUR. OBJ 的目标反汇编程序清单。您会看出它们是一致的。

程序 6－14. 原始的 MVCUR. OBJ 的目标反汇编程序清单

```
Module: mvcur1.asm

segment: 'TEXT'  BYTE USE32 00000015 bytes
0000 55              -MVCUR   push ebp
0001 8b ec                    mov ebp,esp
0003 b4 0f                    mov ah,0fh
0005 cd 10                    int 10h
0007 8a 75 08                 mov dh,+8h[ebp]
000a 8a 55 0c                 mov dl,+0ch[ebp]
000d b4 02                    mov ah,02h
000f cd 10                    int 10h
0011 8b e5                    mov esp,ebp
0013 5d                       pop ebp
0014 c3                       ret

No disassembly errors
```

程序 6—15. 更新的 MVCUR. OBJ 的目标反汇编程序清单

```
Module: mvcur. asm

Group: ' DGROUP'  -DATA

Segment: ' TEXT'  DWORD 00000015 bytes
0000  55          —mvCur  push ebp
0001  8b ec               mov ebp,esp
0003  b4 0f               mov ah,0fh
0005  cd 10               int 10h
0007  8a 75 08            mov dh,+8h[ebp]
000a  8a 55 0c            mov dl,+0ch[ebp]
000d  b4 02               mov ah,02h
000f  cd 10               int 10h
0011  8b e5               mov esp,ebp
0013  5d                  pop ebp
0014  c3                  ret

No disassembly errors
```

下面再看一看,用汇编形式来代替函数 mvCur(...),给 HELLO3. EXP 文件长度造成的影响。

| 程 序 名 称 | 文 件 大 小 | 参数传递方式 |
|---|---|---|
| HELLO2. EXP | 7,948 字节 | 堆 栈 |
| HELLO3. EXP | 7,500 字节 | * 寄存器 |
| HELLO3. EXP | 6,364 字节 | * 寄存器、汇编形式 mvCur |

* 函数 mvCur(...)取参数是通过堆栈

依靠编译器选项,使函数从 C 转到汇编到底能省多大空间、提高多少性能,没必要具

体计算。我们仅仅把C代码的 mvCur(...) 转化为汇编形式的 mvCur(...)就节省了不止1K 字节。假如在一个程序中有500甚至更多个此类函数呢,用C代码、还是汇编代码?哪个程序更精练,不言而喻!

# 第七章　库函数头文件

内容提要

- 头文件 TPROTO. H
- 头文件 TSTRUCT. H
- 头文件 KEYBOARD. H
- 头文件 ASCII. H

使用头文件是众多程序员都非常熟悉的编程技术，它能使定义、函数样板、结构等等标准化。关于使用头文件可以明确的一点是，每次写一个新程序时，不必连续地定义标号、结构等。使用头文件的另一个特点是，可以使某些定义、结构个体化，以符合将一个源文件移植到另一编程环境的要求。

现在逐一介绍四个头文件：

TPROTO. H

TSTRUCT. H

KEYBOARD. H

ASC II. H

## 第一节　头文件 TPROTO. H

这个头文件用来为 TAB 库、和"C"erious 386PM 库的函数定义样板。注意，此处不同于 16 位编译器的情形，使用了两个新参数来定义函数样板，这些参数用来定义汇编函数并且告诉 C 编译器这些函数使用 Microsoft C 的堆栈参数传递约定。

用一个汇编语言模块 mvCur(...) 作例子，有两种方法通知编译器此函数使用堆栈参数传递。第一个方法，用 CDECL 来说明函数：

1. 缺省值表示 寄存器参数传递，如

```
void mvCur(int,int);
```

2. cdecl 不缺省表示 堆栈参数传递，如

```
void cdecl mvCur(int,int);
```

cdecl 也告诉 C 编译器，函数名字前将包含一个前下划线。Watcom C 使用后下划线。当函数名出现在目标模块里面时，Microsoft C 、Watcom C 和 High C 编译器对函数名控制如下：

| Microsoft C | Watcom C | High C |
| --- | --- | --- |
| -mvCur | mvCur- | mvCur |

只能在您的汇编语言模块和 Microsoft 兼容时，才能用 CDECL 。这意味着函数名含前下划线而且使用堆栈参数传递。如果您的汇编语言模块要在一个不同的编译器下工作，那您必须使用强有力的编译指示语句，通知 Watcom C v8. 0 如何处理函数名称以及参数传递。

编译指示语句允许您为任何函数生成特别的参数传递。这表明这个编译器可以定义任何命名约定以及任何参数传递约定。如果您计划为 Watcom C v8. 0 编写高度优化的库程序，那您可以将 C 函数、汇编函数与纯汇编语言通过界面连接起来。

对 Watcom C 强大的编译指示语句的所有 I/O 操作的描述，已经超出了本书重点。不过，这也许能帮助 Watcom C v8. 0 用户明白如何正确书写编译指示。在 Watcom C v8. 0 的头文件 STDDEF. H 中，cdecl 用下述编译指示来定义：

```
#pragma aux cdecl "-*" parm caller[] \
value struct float struct routine [EAX]\
modify [eax ebx ecx edx];
```

如果您决定重新定义函数样板的编译指示方法，应按下述方式去作：

```
    ·
void mvCur (int,int);
    ·
#pragma aux (cdecl) mvCur;
    ·
```

看到函数 mvCur 已经按标准形式形成样板，然后用前面定义的 cdecl 编译指示来修改样板定义，以告知 Watcom C v8. 0 函数 mvCur 是 Microsoft 兼容的。

现在您有些汇编生成的目标代码，它们是为 MetaWare 的 High C 编译器编写的，怎么办？我们要做的所有事情是定义一个编译指示，以描述 High C 编译器的命名约定和参数传递约定，然后用新的 High C 编译指示重新定义函数样板，以取代用 cdecl 编译指示作过的定义。下面是一个 High C 编译指示的例子：

```
#pragma aux H-C "*" \
        parm caller[] \
        value no8087 \
        modify [eax ecx edx fs gs];
```

函数 mvCur 的说明，以及之后把 mvCur 重新定义成 High C 兼容函数，如下：

```
    ·
void mvCur(int,int);
    ·
#pragma aux (H-C) mvCur;
```

Watcom C v8.0 版本的 TPROTO.H 清单见程序 7—1.

程序 7—1. TPROTO.H 清单

```
///////////////////////////////
// TPROTO.H                 //
// function prototypes      //
///////////////////////////////
//include files here
#include <keyboard.h>
#include <ascii.h>
#include <tstruct.h>
#include <stddef.h>

//device header structure
struct devhdr
{
      long dh-next;
      short dh-attr;
      unsigned short dh-strat;
      unsigned short dh-inter;
      char dh-name[8];
}
//////////////////////////////////
// Disk Routines                //
//////////////////////////////////
long          diskTree(void);
DSKINFO       *diskInfo(DSKINFO *);
int           getdrive(void);
void          setdrive(int);

//////////////////////////////////
// string Routines              //
//////////////////////////////////
long      atob(char *);
long      atoh(char *);
void      delChar(char *);
int       findChar(int,char *);
```

```
void      insChar(char * ,char);
void      insNum(char * ,int);
char      * memichar(char * ,char,int);
char      * ritoa(int,char * ,int,int,char);
void      strCjust(char * );
char      * strins(char * ,char * ,int);
void      stripblk(char * );
void      strEnul(char * );
void      strLjust(char * );
void      strRjust(char * );
int       strAnal(int * ,int * ,char * );
char      * words(long,char * );
char      * word0-19(int);
unsigned  char  crotl(unsigned char,int);
unsigned  char  crotr(unsigned char,int);

////////////////////////////////
// Keyboard Routines          //
////////////////////////////////
void      capLkoff();
void      capLkon();
char      gtChar(void);
int       gtKey(void);
char      gtScan(void);
int       gtKBstat(void);
int       gtKBflag(void);
int       gtKBflsh(int);
int       inpflt(float * ,int);
int       inpnum(long * ,int);
void      insoff(void);
void      inson(void);
int       kbstuff(char);
void      numLkoff(void);
void      numLkon(void);
int       prompt(char * ,int);
int       promptne(char * ,int);
void      scrLkoff(void);
void      scrLkon(void);
```

```
/////////////////////////////////
// Cursor Routines            //
/////////////////////////////////
void      gtCur(int *,int *);
int       g-shape(void);
void      mvCur(int,int);
void      rmvCur(int,int);
void      offCur(void);
void      onCur(void);
void      putCR(void);
void      putLF(void);
void      putCRLF(void);
void      ssizeCur(void);
void      rsizeCur(void);
void      sizeCur(void);
void      sCloc(int,int);
void      s-shape(int);
void      rCloc(void);
void      wmvCur(WIND *,int,int);

/////////////////////////////////
// Rectangle Routines          //
/////////////////////////////////
RECT        *setRect(RECT *,int,int,int,int);
void        addRect(RECT *,RECT *);
void        subRect(RECT *,RECT *);
void        dupRect(RECT *,RECT *);
void        dsyRect(RECT *);
void        offRect(RECT *,int,int);
void        boxRect(RECT *,int,int);
void        shftRect(RECT *,int,int);
void        expdRect(RECT *,int,int);
void        ciroffRect(RECT *);
void        fillRect(RECT *,int);
void        saveRect(RECT *);
unsigned    int sizeRect(RECT *);
void        restRect(RECT *);
```

```
void        scUp(RECT * ,int,int);
void        scDN(RECT * ,int,int);
void        cdei aclrrect(int,int,int,int,int);

////////////////////////////////
// Sound Routines             //
////////////////////////////////
void        beep(void);
void        bleep(void);
void        offSound(void);
void        onSound(int);

////////////////////////////////
// Print Routines             //
////////////////////////////////
int         prChar(int,char);
int         print(int);
int         prScrn(int);
int         prScrnFF(int);
int         prStatus(int);
int         psadd(void);
int         pscan(void);
void        far    * psqueue(void);
int         psremv(int);
int         psrestrt(void);
int         psstat(void);

////////////////////////////////
// Screen Routines            //
////////////////////////////////
void        ascup(int,int,int,int,int,int,int);
void        bRScrn(void);
void        bSScrn(void);
int         button(char * * ,char * * ,int,int,char * ,int,int);
int         buttonx(char * * ,char * * ,char * * ,int,int,char * ,int,int);
void        clrpage(int,int);
void        copymono(void);
void        copypage(int,int);
```

```
void    dialog(char ** ,int,int,char * ,int);
int     dropmenu(char ** ,int,int);
int     drpmsmnu(char ** ,int,int);
int     funckeys(char ** ,int);
int     intense(int);
int     inverse(int);
void    *promenu(char ** ,void ** ,int,int,char * ,int,int);
void    putChr(char);
void    putCRLF(void);
void    putLF(void);
void    putCR(void);
void    putStr(char * );
int     cdecl scrdChar(int,int);
void    vdAttr(int,int,int,int);
void    vdChar(int,int,int);
void    vdedit(char * ,int,int,int,int,int);
void    vdHoriz(int,int,int,int);
void    vdpATTR(int,int,int,int,int);
void    vdpChar(int,int,int,int);
void    vdpChr(int,int,int);
void    vdpHoriz(int,int,int,int,int);
int     vdprmne(char * ,int,int,int,int);
int     vdprompt(char * ,int,int,int,int);
void    vdpVert(int,int,int,int,int);
void    vdpWrite(int,int,int,int,char * ,int);
void    vdStr(int,int,int,char * ,char);
void    vdVert(int,int,int,int);
void    vdWrite(int,int,int,char * ,int);
void    viInit(void);
void    vrdChar(int,int);
void    wrChar(char,int);
void    saveScrn(void);
void    resrScrn(void);

void    cdecl scToken(short int);
void    cdecl scChar(short int,short int,short int);
void    cdecl scChr(short int,short int,char);
//void    cdecl scWrite(short int,short int,short int,char * ,short int);
```

```
//void    cdecl scAttr(short int,short int,short int,short int);

//////////////////////////////////
// Window Routines              //
//////////////////////////////////
void    clrWind(WIND *);
void    dispWind(WIND *);
void    dsyWind(WIND *);
void    rdImg(WIND *);
void    rdWind(WIND *);
void    remvWind(WIND *);
void    setAttr(WIND *,int);
void    setBoard(WIND *,int);
void    setTitle(WIND *,char *);
void    *setWind(WIND *,int,int,int,int);
void    strtWind(WIND *);
unsigned    int    sizeImag(WIND *);
void    wmvCur(WIND *,int,int);
void    wrBox(WIND *);
void    wrImg(WIND *);
void    wrWind(WIND *);
void    wvdAttr(WIND *,int,int,int,char);
void    wvdhoriz(WIND *,int,int,int,int);
int     wvdprmpt(WIND *,char,int,int,int,char);
int     wvdprmpt(WIND *,char,int,int,int,int);
void    wvdVert(WIND *,int,int,int,int);
void    wvdScdn(WIND *,int);
void    wvdScup(WIND *,int);
void    wvdStr(WIND *,int,int,int,char *,char);
void    wvdWrite(WIND *,int,int,int,char *,char);
int     wvrdChar(WIND *,int,int);

//////////////////////////////////
// New Window Routines          //
//////////////////////////////////
void    clrtWind(TWIND *);
void    dispWin(TWIND *);
void    remvtWin(TWIND *);
```

```
void    settAttr(TWIND * ,int);
void    settBord(TWIND * ,int);
void    settTitl(TWIND * ,char * );
TWIND    * settWin(TWIND * ,int,int,int,int,unsigned int * );
void    strttWind(TWIND * );
void    tvdAttr(TWIND * ,int,int,int,char);
void    tvdHoriz(TWIND * ,int,int,int,int);
void    tvdVert(TWIND * ,int,int,int,int);
void    tvdWrite(TWIND * ,int,int,int,char * ,char);
int     tvrdChar(TWIND * ,int,int);

//////////////////////////////////
// Miscellaneous Routines       //
//////////////////////////////////
unsigned   long   bitfld(unsigned long,int,int);
int      bitflds(unsigned long,int,int,char * );
void     Delay(int,int);
int      isqrt(long);
int      mkToken(int,int);
int      mkAttr(int,int,int,int);
int      numpport(void);
int      numsport(void);
void     setpage(int);
void     smbits(int,int,char * ,int);
void     smbitsv(int,int,unsigned long,int,int);
void     sudchar(int,int,int,int);
void     ascup(int,int,int,int,int,int,int);
void     gtMode(int * ,int * ,int * );
void     rLpen(LIGHT-PEN * );
int      ramSize(void);
void     exit-bad(char * );

void     DispErr(void);
void     ErasRee(void);

//calling convention pragmas tell
//copiler that these routines use
//Microsoft naming and on register
```

```
// calling conventions

#pragma      aux    (cdecl)    vidInit;

#pragma      aux    (cdecl)    diskFree;
#pragma      aux    (cdecl)    diskInfo;
#pragma      aux    (cdecl)    getDrive;
#pragma      aux    (cdecl)    setdrive;

#pragma      aux    (cdecl)    atob;
#pragma      aux    (cdecl)    atoh;
#pragma      aux    (cdecl)    delChar;
#pragma      aux    (cdecl)    findChar;
#pragma      aux    (cdecl)    insChar;
#pragma      aux    (cdecl)    insNum;
#pragma      aux    (cdecl)    memichr;
#pragma      aux    (cdecl)    msinit;
#pragma      aux    (cdecl)    msstat;
#pragma      aux    (cdecl)    mssetxy;
#pragma      aux    (cdecl)    msstaeh1;
#pragma      aux    (cdecl)    msstaeh2;
#pragma      aux    (cdecl)    msstaeh3;
#pragma      aux    (cdecl)    msvlim;
#pragma      aux    (cdecl)    mssudeh;
#pragma      aux    (cdecl)    msstpage;
#pragma      aux    (cdecl)    msstirat;
#pragma      aux    (cdecl)    msstgcur;
#pragma      aux    (cdecl)    msstexar;
#pragma      aux    (cdecl)    mssetmpr;
#pragma      aux    (cdecl)    mssavetsr;
#pragma      aux    (cdecl)    mssavems;
#pragma      aux    (cdecl)    mssave;
#pragma      aux    (cdecl)    msrmc;
#pragma      aux    (cdecl)    msrestms;
#pragma      aux    (cdecl)    msreleas;
#pragma      aux    (cdecl)    mspress;
#pragma      aux    (cdecl)    mson;
#pragma      aux    (cdecl)    msoff;
```

```
# pragma     aux    (cdecl)    mslpon;
# pragma     aux    (cdecl)    mslpoff;
# pragma     aux    (cdecl)    msinfo;
# pragma     aux    (cdecl)    mshlim;
# pragma     aux    (cdecl)    msgtsens;
# pragma     aux    (cdecl)    msgtpage;
# pragma     aux    (cdecl)    msgsbufs;
# pragma     aux    (cdecl)    msgaeh;
# pragma     aux    (cdecl)    ritoa;
# pragma     aux    (cdecl)    strCjust;
# pragma     aux    (cdecl)    strins;
# pragma     aux    (cdecl)    stripblk;
# pragma     aux    (cdecl)    strEnul;
# pragma     aux    (cdecl)    strLjust;
# pragma     aux    (cdecl)    strRjust;
# pragma     aux    (cdecl)    strAnal;
# pragma     aux    (cdecl)    words;
# pragma     aux    (cdecl)    word0-19;
# pragma     aux    (cdecl)    crotl;
# pragma     aux    (cdecl)    crotr;

# pragma     aux    (cdecl)    caplkoff;
# pragma     aux    (cdecl)    caplkon;
# pragma     aux    (cdecl)    gtChar;
# pragma     aux    (cdecl)    gtKey;
# pragma     aux    (cdecl)    gtScan;
# pragma     aux    (cdecl)    gtKBstat;
# pragma     aux    (cdecl)    gtKBflag;
# pragma     aux    (cdecl)    gtKBflsh;
# pragma     aux    (cdecl)    inpflt;
# pragma     aux    (cdecl)    inpnum;
# pragma     aux    (cdecl)    insoff;
# pragma     aux    (cdecl)    inson;
# pragma     aux    (cdecl)    kbstuff;
# pragma     aux    (cdecl)    numlkoff;
# pragma     aux    (cdecl)    numlkon;
# pragma     aux    (cdecl)    prompt;
# pragma     aux    (cdecl)    promptne;
```

```
# pragma    aux    (cdecl)    scrlkoff;
# pragma    aux    (cdecl)    scrlkon;

# pragma    aux    (cdecl)    gtCur;
# pragma    aux    (cdecl)    g-shape;
# pragma    aux    (cdecl)    mvCur;
# pragma    aux    (cdecl)    offCur;
# pragma    aux    (cdecl)    onCur;
# pragma    aux    (cdecl)    putCR;
# pragma    aux    (cdecl)    putLF;
# pragma    aux    (cdecl)    putCRLF;
# pragma    aux    (cdecl)    ssizeCur;
# pragma    aux    (cdecl)    rsizeCur;
# pragma    aux    (cdecl)    sizeCur;
# pragma    aux    (cdecl)    sCloc;
# pragma    aux    (cdecl)    s-shape;
# pragma    aux    (cdecl)    rCloc;
# pragma    aux    (cdecl)    wmvCur;
# pragma    aux    (cdecl)     * setRect;
# pragma    aux    (cdecl)    addRect;
# pragma    aux    (cdecl)    subRect;
# pragma    aux    (cdecl)    dupRect;
# pragma    aux    (cdecl)    dsyRect;
# pragma    aux    (cdecl)    offRect
# pragma    aux    (cdecl)    boxRect;
# pragma    aux    (cdecl)    clrRect;
# pragma    aux    (cdecl)    fillRect
# pragma    aux    (cdecl)    saveRect
# pragma    aux    (cdecl)    sizeRect;
# pragma    aux    (cdecl)    restRect;
# pragma    aux    (cdecl)    scUp;
# pragma    aux    (cdecl)    scDn;

# pragma    aux    (cdecl)    beep;
# pragma    aux    (cdecl)    bleep;
# pragma    aux    (cdecl)    offsound;
# pragma    aux    (cdecl)    onSound;
```

```
# pragma    aux    (cdecl)    prChar;
# pragma    aux    (cdecl)    prInit;
# pragma    aux    (cdecl)    prScrn;
# pragma    aux    (cdecl)    prScrnFF;
# pragma    aux    (cdecl)    prStatus;
# pragma    aux    (cdecl)    psadd;
# pragma    aux    (cdecl)    pscan;
# pragma    aux    (cdecl)    psqueue;
# pragma    aux    (cdecl)    psremv;
# pragma    aux    (cdecl)    psrestrt;
# pragma    aux    (cdecl)    psstat;

# pragma    aux    (cdecl)    ascup;
# pragma    aux    (cdecl)    bRScrn;
# pragma    aux    (cdecl)    bSScrn;
# pragma    aux    (cdecl)    button;
# pragma    aux    (cdecl)    buttonx;
# pragma    aux    (cdecl)    clrpage;
# pragma    aux    (cdecl)    copymono;
# pragma    aux    (cdecl)    copypage;
# pragma    aux    (cdecl)    dialog;
# pragma    aux    (cdecl)    dropmenu;
# pragma    aux    (cdecl)    drpmsmnu;
# pragma    aux    (cdecl)    funcKeys;
# pragma    aux    (cdecl)    ense;
# pragma    aux    (cdecl)    inverse;
# pragma    aux    (cdecl)    popmenu;
# pragma    aux    (cdecl)    putChr;
# pragma    aux    (cdecl)    putCRLF;
# pragma    aux    (cdecl)    putLF;
# pragma    aux    (cdecl)    putCR;
# pragma    aux    (cdecl)    putStr;
# pragma    aux    (cdecl)    rdChar;
# pragma    aux    (cdecl)    scrnClr;
# pragma    aux    (cdecl)    vdAttr;
# pragma    aux    (cdecl)    vdChar;
# pragma    aux    (cdecl)    vdChr;
# pragma    aux    (cdecl)    vdedit;
```

```
#pragma aux (cdecl) vdHoriz;
#pragma aux (cdecl) vdpATTR;
#pragma aux (cdecl) vdpChar;
#pragma aux (cdecl) vdpChr;
#pragma aux (cdecl) vdpHoriz;
#pragma aux (cdecl) vdprmne;
#pragma aux (cdecl) vdprompt;
#pragma aux (cdecl) vdpVert;
#pragma aux (cdecl) vdpWrite;
#pragma aux (cdecl) vdStr;
#pragma aux (cdecl) vdVert;
#pragma aux (cdecl) vdWrite;
#pragma aux (cdecl) vidInit;
#pragma aux (cdecl) vrdChar;
#pragma aux (cdecl) wrChar;
#pragma aux (cdecl) clrWind;
#pragma aux (cdecl) dispWind;
#pragma aux (cdecl) dsyWind;
#pragma aux (cdecl) rdImg;
#pragma aux (cdecl) rdWind;
#pragma aux (cdecl) remvWind;
#pragma aux (cdecl) setAttr;
#pragma aux (cdecl) setBoard;
#pragma aux (cdecl) setTitle;
#pragma aux (cdecl) setWind;
#pragma aux (cdecl) sizeImg;
#pragma aux (cdecl) wmvCur;
#pragma aux (cdecl) wrBox;
#pragma aux (cdecl) wrImg;
#pragma aux (cdecl) wrWind;
#pragma aux (cdecl) wrdAttr;
#pragma aux (cdecl) wvdHoriz;
#pragma aux (cdecl) wvdprmne;
#pragma aux (cdecl) wvdprmpt;
#pragma aux (cdecl) wvdVert;
#pragma aux (cdecl) wvdScdn;
#pragma aux (cdecl) wvdScup;
#pragma aux (cdecl) wvdStr;
```

```
# pragma   aux   (cdecl)   wvrdChar;
# pragma   aux   (cdecl)   clrtWind;
# pragma   aux   (cdecl)   disptWin;
# pragma   aux   (cdecl)   remvtWin;
# pragma   aux   (cdecl)   settAttr;
# pragma   aux   (cdecl)   settBord;
# pragma   aux   (cdecl)   settTitl;
# pragma   aux   (cdecl)   settWin;
# pragma   aux   (cdecl)   strttWin;
# pragma   aux   (cdecl)   tvdAttr;
# pragma   aux   (cdecl)   tvdHoriz;
# pragma   aux   (cdecl)   tvdVert;
# pragma   aux   (cdecl)   tvdWrite;
# pragma   aux   (cdecl)   tvrdChar;

# pragma   aux   (cdecl)   bitfld;
# pragma   aux   (cdecl)   bitflds;
# pragma   aux   (cdecl)   Delay;
# pragma   aux   (cdecl)   isqrt;
# pragma   aux   (cdecl)   mkToken;
# pragma   aux   (cdecl)   mkAttr;
# pragma   aux   (cdecl)   numpport;
# pragma   aux   (cdecl)   numsport;
# pragma   aux   (cdecl)   smbits;
# pragma   aux   (cdecl)   smbitsv;
# pragma   aux   (cdecl)   sud;
# pragma   aux   (cdecl)   ascup;
# pragma   aux   (cdecl)   gtmode;
# pragma   aux   (cdecl)   rLpen;
# pragma   aux   (cdecl)   ramSize;
# pragma   aux   (cdecl)   exit-bad;
# pragma   aux   (cdecl)   setpage;

# pragma   aux   (cdecl)   getvec;
# pragma   aux   (cdecl)   setvec;
# pragma   aux   (cdecl)   setCE;

# pragma   aux   (cdecl)   DispErr;
```

```
#pragma        aux    (cdecl)    ErasErr;
//
//End of TPROTO.h
//////////////////////////////////////////////
```

## 第二节 头文件 TSTRUCT.H

程序 7—2.给出了 TSTRUCT.H 清单。这里的结构定义选择了 32 位整数,并且修改了汇编模块,这都是为了适应 16 位整数向 32 位整数的转化。

程序 7—2. TSTRUCT.H 清单

```
////////////////////////////////////////
// TSTRUCT.H                        //
// Structres and Function Defines   //
////////////////////////////////////////

//general purpose rectangle structre and defines
#define    IMAGE unsigned int

////////////////////////////////////////////////////////////
// WIND structre                                          //
// Description:                                           //
//         Structre which holds parameters of window.     //
////////////////////////////////////////////////////////////
typedef struct
{
      int   ul-row;
      int   ul-col;
      int   lr-row;
      int   lr-col;
      unsigned int img-size;
      unsigned int far  * img-ptr;
      unsigned int far  * wind-ptr;
      int   box-type;
      int   attr;
      int   visible;
      int   top-offset;
```

```
        int   top-length;
        int   show-top;
        int   bot-offset;
        int   bot-length;
        int   show-bot;
    char  * t-title;
    char  * b-title;
    }WIND;

    ////////////////////////////////////////////////////////////////
    // TWIND structure                                            //
    // Description:                                               //
    //       Structure which holds parameters of windows          //
    ////////////////////////////////////////////////////////////////
    typedef struct
    {
        int       ul-row;
        int       ul-col;
        int       lr-row;
        int       lr-col;
        unsigned int  * img-ptr;
        int       box-type;
        int       attr;
        int       visible;
        int       show-top;
        int        * t-title;
    }TWIND;

    ////////////////////////////////////////////////////////////////
    // DSKINFO structure                                          //
    // Description:                                               //
    //      Structure which holds disk information                //
    ////////////////////////////////////////////////////////////////
    typedef stuct
    {
        unsigned char media-descr;
        unsigned int clust-avail;
        unsigned int clust-total;
```

```
    unsigned int sec-p-clust;
    unsigned int bytes-p-sec;
}VIDEO;

////////////////////////////////
// RECT structure definition   //
////////////////////////////////
typedef struct
{
    int  ul-row;
    int  ul-col;
    int  lr-row;
    int  lr-col;
    unsigned   int * image;
}RECT;

//////////////////////////////////
// cur-LOCATION structure       //
// Description: Cursor Location //
//////////////////////////////////
typedef struct
{
     int      row;
     int      column;
}CUR-LOCATION;

//////////////////////////////////////////////////////
//LIGHT-PEN structure                               //
//Description:                                      //
//     structure which holds light pen information  //
//////////////////////////////////////////////////////
typedef struct
{
    int  status;
    int  pix-col;
    int  pix-row1;
    int  pix-row2;
    int  ch-row;
```

```
    int   ch-col;
}LIGHT-PEN;

///////////////////////////
// Defines for WRBOX     //
///////////////////////////
#define   S-S-S-S    0
#define   S-S-D-D    0
#define   D-D-S-S    0
#define   D-D-D-D    0

///////////////////////////
// Defines for MKATTR    //
///////////////////////////
#define   BLACK              0
#define   BLUE               1
#define   GREEN              2
#define   CYAN               3
#define   RED                4
#define   MAGENT             5
#define   BROWN              6
#define   WHITE              7
#define   NORMAL             7
#define   REVERSE            112

#define   ON-INTENSITY       8
#define   OFF-INTENSITY      0
#define   ON-BLINK           128
#define   OFF-BLINK          0

///////////////////////////
// Defines for scroll    //
///////////////////////////
#define   UP-SCROLL       6
#define   DOWN-SCROLL     7

///////////////////////////
// Defines for printer   //
```

```
////////////////////////////
#define    PRINT-TIME-OUT          1
#define    IO-ERROR                4
#define    PRINT-SELECTED          8
#define    OUT-OF-PAPER            16
#define    ACKNOWLEDGE             32
#define    PRINT-NOT-BUSY          64

////////////////////////////////////////////////
// Defines for flush kb buffer and & char    //
////////////////////////////////////////////////
#define    ON-ECHO-CTRL              1
#define    OFF-ECHO-CTRL-C           7
#define    OFF-ECHO                  8

////////////////////////////////////
// Defines for kb shift status    //
////////////////////////////////////
#define RIGHT-SHIFT         1
#define LEFT-SHIFT          2
#define CTRL-PRESS          4
#define ALT-PRESS           8
#define SCROLL-LOCK         16
#define NUM-LOCK            32
#define CAPS-LOCK           64
#define INSERT-ON           128

//////////////////////////////////
// Defines for MENU routions    //
//////////////////////////////////
#define CENTER         0xFF
#define NUMBERED       1
#define RESETROW       2

//////////////////////////////
// Defines for vdEdit       //
//////////////////////////////
#define UPPER          1
```

```
#define LOWER        2
#define NAME         3

//////////////////////////////////
// Defines for mouse routines   //
//////////////////////////////////
#define    LEFTBUTTON      1
#define    RIGHTBUTTON     2
#define    CNTRBUTTON      4
//
// End of TSTRUCT.H
//////////////////////////////////
```

## 第三节　头文件 KEYBOARD. H

KEYBOARD. H(程序 7—3.)定义了很多键以及键盘组合，这都是为了方便特别键盘控制器的开发。

程序 7—3. KEYBOARD. H 清单

```
//////////////////////////////////////////
// KEYBOARD.H                           //
// Description:                         //
// Defintions of keyboard 16bit codes   //
// Included in TPROTO.H                 //
//////////////////////////////////////////
// soecial key codes
#define    INSERT              0X5200
#define    DELETE              0X5300
#define    SPACE               0X3920
#define    ESC                 0X011B
#define    ESCAPE              0X011B
#define    PGDN                0X5100
#define    PGUP                0X4900
#define    PERIOD              0X342E
#define    TAB                 0X0F09
#define    RT-SQUARE           0X1B5D
#define    LT-SQUARE           0X1A5B
```

```
#define   RT-BRACKET       0X1B7D
#define   LT-BRACKET       0X1A7B
#define   CNTL-HOME        0X7700
#define   CNTL-END         0X7500
#define   CNTL-ENTER       0X1C0A
#define   CNTL-BS          0X0E7F
#define   HOME             0X4700
#define   END              0X4F00
#define   s-BS             0X0008
#define   BS               0X0E08
#define   BACKSPACE        0X0E08
#define   s-CR             0X000D
#define   CR               0X1C0D
#define   ENTER            0X1C0D
#define   UP-ARROW         0X4800
#define   RIGHT-ARROW      0X4D00
#define   LEFT-ARROW       0X4B00
#define   DOWN-ARROW       0X5000

// function key codes

#define   F1               0X3b00
#define   F2               0X3c00
#define   F3               0X3d00
#define   F4               0X3e00
#define   F5               0X3f00
#define   F6               0X4000
#define   F7               0X4100
#define   F8               0X4200
#define   F9               0X4300
#define   F10              0X4400

// shift special keys

#define   SHIFT-TAB        0X3c00
#define   SHIFT-HOME       0X4737
#define   SHIFT-END        0X4f31
#define   SHIFT-INSERT     0X5230
```

```
#define   SHIFT-DELETE          0X532e
#define   SHIFT-INSERT          0X5230

// shift function keys

#define   SHFT-F1               0X5400
#define   SHFT-F2               0X5500
#define   SHFT-F3               0X5600
#define   SHFT-F4               0X5700
#define   SHFT-F5               0X5800
#define   SHFT-F6               0X5900
#define   SHFT-F7               0X5A00
#define   SHFT-F8               0X5B00
#define   SHFT-F9               0X5C00
#define   SHFT-F10              0X5D00

// shift arrow keys

#define   SH-R-ARROW            0X4d36
#define   SH-L-ARROW            0X4b34
#define   SH-U-ARROW            0X4838
#define   SH-D-ARROW            0X5032

// control function keys

#define   CNTL-F1               0X5e00
#define   CNTL-F2               0X5f00
#define   CNTL-F3               0X6000
#define   CNTL-F4               0X6100
#define   CNTL-F5               0X6200
#define   CNTL-F6               0X6300
#define   CNTL-F7               0X6400
#define   CNTL-F8               0X6500
#define   CNTL-F9               0X6600
#define   CNTL-F10              0X6700
#define   CNTL-LEFTA            0X7300
#define   CNTL-RIGHTA           0X7400
```

```
// alt function keys

#define   ALT-F1          0X6800
#define   ALT-F2          0X6900
#define   ALT-F3          0X6A00
#define   ALT-F4          0X6B00
#define   ALT-F5          0X6C00
#define   ALT-F6          0X6D00
#define   ALT-F7          0X6E00
#define   ALT-F8          0X6F00
#define   ALT-F9          0X7000
#define   ALT-F10         0X7100

// alt alpha keys

#define   ALT-A           0X1e00
#define   ALT-B           0X3000
#define   ALT-C           0X2e00
#define   ALT-D           0X2000
#define   ALT-E           0X1200
#define   ALT-F           0X2100
#define   ALT-G           0X2200
#define   ALT-H           0X2300
#define   ALT-I           0X1700
#define   ALT-J           0X2400
#define   ALT- K          0X2500
#define   ALT-L           0X2600
#define   ALT-M           0X3200
#define   ALT-N           0X3100
#define   ALT-O           0X1800
#define   ALT-P           0X1900
#define   ALT-Q           0X1000
#define   ALT-R           0X1300
#define   ALT-S           0X1f00
#define   ALT-T           0X1400
#define   ALT-U           0X1600
#define   ALT-V           0X2f00
#define   ALT-W           0X1100
```

```
#define    ALT-X                    0X2d00
#define    ALT-Y                    0X1500
#define    ALT-Z                    0X2c00

// control alpha keys

#define    CNTL-A                   0X1e01
#define    CNTL-B                   0X3002
#define    CNTL-C                   0X2e03
#define    CNTL-D                   0X2004
#define    CNTL-E                   0X1205
#define    CNTL-F                   0X2106
#define    CNTL-G                   0X2207
#define    CNTL-H                   0X2308
#define    CNTL-I                   0X1709
#define    CNTL-J                   0X240a
#define    CNTL-K                   0X250b
#define    CNTL-L                   0X260c
#define    CNTL-M                   0X320d
#define    CNTL-N                   0X310e
#define    CNTL-O                   0X180f
#define    CNTL-P                   0X1910
#define    CNTL-Q                   0X1011
#define    CNTL-R                   0X1312
#define    CNTL-S                   0X1f13
#define    CNTL-T                   0X1414
#define    CNTL-U                   0X1615
#define    CNTL-V                   0X2f16
#define    CNTL-W                   0X1117
#define    CNTL-X                   0X2d18
#define    CNTL-Y                   0X1519
#define    CNTL-Z                   0X2c1a

//numbers

#define    K-0                      0X0b30
#define    K-1                      0X0231
#define    K-2                      0X0332
```

```
#define   K-3          0X0433
#define   K-4          0X0534
#define   K-5          0X0635
#define   K-6          0X0736
#define   K-7          0X0837
#define   K-8          0X0938
#define   K-9          0X0a39

// alt numbers

#define   ALT-0        0X8100
#define   ALT-1        0X7800
#define   ALT-2        0X7900
#define   ALT-3        0X7a00
#define   ALT-4        0X7b00
#define   ALT-5        0X7c00
#define   ALT-6        0X7d00
#define   ALT-7        0X7e00
#define   ALT-8        0X7f00
#define   ALT-9        0X8000

// more special keys

#define   K-SPACE      0X3920
#define   K-EXCLAN     0X0221
#define   K-QUOTE      0X2822
#define   K-POUND      0X0423
#define   K-DOLLAR     0X0524
#define   K-PERCENT    0X0625
#define   K-AND        0X0826
#define   K-APOST      0X2827
#define   K-LPAREN     0X0A28
#define   K-RPAREN     0X0B29
#define   K-STARE      0X092A
#define   K-PLUS       0X0D2B
#define   K-COMMA      0X332C
#define   K-MINUS      0X0C2D
#define   K-PERIOD     0X342E
```

```
#define    K-FSLASH                    0X352F
#define    K-COLON                     0X273A
#define    K-SCOLON                    0X273B
#define    K-LESS                      0X333C
#define    K-EQUAL                     0X0D3D
#define    K-GREAT                     0X343E
#define    K-QUEST                     0X353F
#define    K-AMPER                     0X0340

//upper case keys

#define    K-A                         0X1e61— 0x20
#define    K-B                         0X3062— 0x20
#define    K-C                         0X2e63— 0x20
#define    K-D                         0X2064— 0x20
#define    K-E                         0X1265— 0x20
#define    K-F                         0X2166— 0x20
#define    K-G                         0X2267— 0x20
#define    K-H                         0X2368— 0x20
#define    K-I                         0X1769— 0x20
#define    K-J                         0X246a— 0x20
#define    K-K                         0X256b— 0x20
#define    K-L                         0X266c— 0x20
#define    K-M                         0X326d— 0x20
#define    K-N                         0X316e— 0x20
#define    K-O                         0X186f— 0x20
#define    K-P                         0X1970— 0x20
#define    K-Q                         0X1071— 0x20
#define    K-R                         0X1372— 0x20
#define    K-S                         0X1f73— 0x20
#define    K-T                         0X1474— 0x20
#define    K-U                         0X1675— 0x20
#define    K-V                         0X2f76— 0x20
#define    K-W                         0X1177— 0x20
#define    K-X                         0X2d78— 0x20
#define    K-Y                         0X1579— 0x20
#define    K-Z                         0X2c7a— 0x20
```

```
// more special keys

#define   K-LBRACK          0X1A5B
#define   K-BSLASH          0X2B5C
#define   K-KBRACK          0X1B5D
#define   K-KARAT           0X075E
#define   K-UNDER           0X0C5C

//lower case keys

#define   K-a               0X1e61
#define   K-b               0X3062
#define   K-c               0X2e63
#define   K-d               0X2064
#define   K-e               0X1265
#define   K-f               0X2166
#define   K-g               0X2267
#define   K-h               0X2368
#define   K-i               0X1769
#define   K-j               0X246a
#define   K-k               0X256b
#define   K-l               0X266c
#define   K-m               0X326d
#define   K-n               0X316e
#define   K-o               0X186f
#define   K-p               0X1970
#define   K-q               0X1071
#define   K-r               0X1372
#define   K-s               0X1f73
#define   K-t               0X1474
#define   K-u               0X1675
#define   K-v               0X2f76
#define   K-w               0X1177
#define   K-x               0X2d78
#define   K-y               0X1579
#define   K-z               0X2c7a
//
//End of KEYBOARD. H
```

```
//////////////////////////////////
```

# 第四节　头文件 ASCII. H

下图向您展示了 ASCII. H，其中包含了一些公共定义。

程序 7—4. ASCII. H 清单

```
//////////////////////////////////////////////
// ASCII. H                                 //
// Description:                             //
// Common ascii descriptions which          //
// use an ' a' prefix. This is done         //
// to prevent duplicate definitions.        //
// This definitions are used in library     //
// files.                                   //
// This files in includeed in TPROTO. H.    //
//////////////////////////////////////////////
#define  aNUL     0    //null \0 delimeter
#define  aSON     1    // —A - start of heading
#define  aSTX     2    // —B - start of text
#define  aETX     3    // —C - end of text
#define  aEOT     4    // —D - end of transmission
#define  aENQ     5    // —E - inpuiry
#define  aACK     6    // —F - affirm acknowledgement
#define  aBEL     7    // —G - audible bell
#define  aBS      8    // —H - backspace
#define  aTAB     9    // —I - horizental tab
#define  aLF      10   // —J - line feed
#define  aVT      11   // —K - vertical tab
#define  aFF      12   // —L - form feed
#define  aCR      13   // —M - carriage return
#define  aSO      14   // —N - shift out
#define  aSI      15   // —O - shift in
#define  aDEC     16   // —P - data link escape
#define  aDC1     17   // —Q - device control 1
#define  aDC2     18   // —R - device control 2
#define  aDC3     19   // —S - device control 3
```

```
#define   aDC4        20     // —T - device control 4
#define   aNAK        21     // —U - neg acknowledege
#define   aSYN        22     // —V - synchronous idel
#define   aETB        23     // —W - end of trensmission
#define   aCAN        24     // —X - cancel
#define   aEM         25     // —Y - end of medium
#define   aSUB        26     // —Z - substitute
#define   aESC        27     //        escape
#define   aFS         28     //        file sererator
#define   aGS         29     //        group seperator
#define   aRS         30     //        record seperator
#define   aUS         31     //        unlinked seperator
#define   aSPC        32     //        space

#define   aCODE       94     // indicates printer command follows
#define   aHCR        aEOT   // Hard carriage return
#define   a           ' C'   // code to center line
#define   a           ' D'   // dounble strike toggle
#define   a           ' E'   // emphasize toggle
#define   a           ' S'   // superscript toggle
#define   a           ' I'   // italics toggle
#define   a           ' B'   // bold toggle

#define   aTRUE       1      // true
#define   aFALSE      0      // false

#define   ONE-COL     1      // 1 column format
#define   TOW-COL     2      // 2 column format
#define   ONE-TOP     3
#define   TOW-TOP     4
#define   ONE-BOT     5
#define   TOW-BOT     6
#define   TOW-LR      7
#define   TOW-R       8
#define   TOW-UR      9
#define   TOW-TB      10
#define   VONE-COL    11     //word per cgart format
#define   XONE-COL    81     //1 column format
```

```
#define  XTOW-COL      82      //2 column format
#define  XTHREE-COL    83      //3 column format
#define  XONE-TOP      84
#define  XTOW-TOP      85
#define  XTHREE-TOP    86
#define  XONE-BOT      87
#define  XTOW-BOT      88
#define  XTHREE-BOT    89
#define  XTHREE-LR     90
#define  XTHREE-R      91
#define  XTHREE-P1     92
#define  XTHREE-P2     93
#define  XTHREE-TB     94
#define  XTHREE-UR     95
#define  XTHREE-2T     96
#define  XTHREE-2B     97
//
// End of ASCII. H
//////////////////////////////////////////////////////
```

概要：

头文件 TPROTO. H、TSTRUCT. H、KEYBOARD. H 和 ASCII. H 已经介绍完毕，后面就该开始实际的建库过程了。

这些文件作出的定义，在本书中将一直使用，它们还被用来快速开发所有的库函数及演示每个库函数性能的一套程序。

# 第八章　库函数遍览

要点

· 光标控制部分
· 屏幕控制部分
· 发声控制部分
· 键盘控制部分
· 矩形块管理部分
· 窗口管理部分
· 鼠标管理部分

在前几章叙述了保护方式、保护方式下的编程，以及库函数的创建，现在就向您展示许多重要的库函数及其应用。它们的功能包含了对光标、屏幕、键盘、声音、鼠标，以及屏幕窗口的操作，而这些都是实际编程中极富魅力的部分。

所有的源程序在第九章中都书写得极其清楚。请注意，在它们能够作为库函数使用之前，必须先对每个源程序进行编译或者汇编，然后再将所得目标代码模块与 TAB. LIB 链接，使之成为 TAB. LIB 库的一员。关于这些操作，简述如下：

若源程序是用 C 语言写成，名为 FUNCTION. C，则进行下列操作

**· 键入命令**

　　CC　FUNCTION

**· 按回车键，生成目标文件 FUNCTION. OBJ，然后，键入命令**

　　WADDLIB · FUNCTION

**· 按回车键，至此完成了目标文件和 TAB. LIB 的链接。**

若源程序由汇编语言写成，且名为 FUNCTION. ASM，则操作如下

**· 键入命令**

　　AMASM　FUNCTION

**· 按回车键，生成目标文件 FUNCTION. OBJ，然后，键入命令**

　　WADDLIB　FUNCTION

**· 按回车键，至此完成了目标文件和 TAB. LIB 的链接。**

上述命令中的“CC”、“AMASM”、“WADDLIB”，是我们为了方便而自行编写的几个批处理文件“CC. BAT”、“AMASM. BAT”、“WADDLIB. BAT”。文件的内容请见 8. 1. 1 节和 8. 1. 2 节。

最后要注意，在演示程序的源代码中，关于大部分函数的语法已经给出了清晰的描

述。在介绍每个函数之后，都用演示程序来形象地表现函数的调用及工作情况。本章提供的演示程序都是专门为适应 Phar Lap 的 DOS－EXTENDER 的工作环境而专门设计的。

总之，这些函数演示程序构成了一整套有效的 TAB 库。某些稍微复杂的函数，在其描述部分都给出了单独的语法描述。为了弄清楚每个函数是如何被调用的，您应该仔细琢磨每个演示程序。

下面逐一介绍每一个函数的具体用法。

## 第一节　光标控制部分

本节内容提要

- 取当前光标位置，gtCur(...)
- 移动光标到指定行列，mvCur(...)
- 移动光标到相对于当前的某个位置，rmvCur(...)
- 保存光标当前位置，sCloc(...)
- 恢复前面保存的光标位置，rCloc(...)
- 关闭光标，offCur(...)
- 打开光标，onCur(...)
- 设置光标大小，sizeCur(...)
- 保存光标当前大小，ssizeCur(...)
- 恢复前面保存的光标大小，rsizeCur(...)

用 BIOS 的 int 10H 的不同功能调用来完成光标控制，非常简单，但为了不超出在此探讨的重点。有关 BIOS 完整的函数清单及相关寄存器的使用约定在此不作介绍，您可以去查阅众多的程序员参考手册。

这一节共列举了 10 个控制光标的函数。

### 8.1.1　取光标当前位置，函数 gtCur(...)

注意，函数 gtCur(...) 使用了 Watcom C v8.0 的函数 int386，与之相对应的是 16 位编译器使用的函数 int86。

对于 gtCur(...)的源代码 GTCUR.C(见第九章)，先用编译—链接程序 WCL386 对其进行编译。在建库过程中，我们编写了一个用于编译—链接的批处理文件 CC.BAT。为把源代码编译成符合库程序要求的模块，就可使用 CC.BAT，此文件很简单，如下：

```
WCL386 /c %1.c
```

开关/c 告诉 WCL386 使用编译器而不使用链接器。您若想编译 GTCUR.C，只需键入：

```
CC GTCUR
```

按 Enter 键。其它事情由 WCL386 编译—链接程序去处理。

为把目标模块加入库中，这里使用了批处理文件 WADDLIB.BAT。WADDLIB.BAT 用于 Watcom 库管理程序。另外还有几个类似功能的批处理文件，用于别的库管理程序，

例如，ADDLIB. BAT 用于 Phar Lap 。把各种库管理程序的繁琐的语法说明都放到各自的批处理文件中，以后凡是遇到使用某个库管理程序，记得它们的批处理文件名即可。

批处理文件 ADDLIB. BAT ，用于 Phar Lap 库管理程序，程序清单如下：

```
386lib \book \lib \tab -ADD %1%2%3%4%5%6
```

批处理文件 WADDLIB. BAT ，用于 Watcom 库管理程序，程序清单如下：

```
wlib \book \lib \tab + %1
```

若准备用 Phar Lap 库管理程序将 GTCUR. OBJ 加入 TAB. LIB ，键入

```
WADDLIB gtCur
```

然后，按 Enter 键。

注意，库管理程序使用 Watcom 链接器和使用 Phar Lap 链接器的语法相差甚远。上述两个批处理文件就清楚地展现了这一点。

至此，已经生成了目标模块 GTCUR. OBJ 并且把它加入了 TAB. LIB 文件中。下面我们测验一下函数 gtCur(...) 的工作情况，利用演示程序 CUR1. C(见程序 8-1-1. )来完成此使命。

程序 8-1-1. 程序 CUR1. C 源代码清单

```
/////////////////////////////////////////
// CUR1. C
//
// Description:
// Test function gtcur(...)
/////////////////////////////////////////

// include files here
#include <stdio. h>
#include <tproto. h>

void main()
{
int row,col;

//get current cursor position
gtcur(&row,&col);

//report cursor position to screen
printf("\nRow= %02d Col= %02d\n",row,col);

}
```

使用 WCL386 命令编译、链接 CUR1. C 极其容易。批处理文件 CCL. BAT ，可用来调用编译一链接应用程序。CCL. BAT 文件如下：

WCL386 %1 c:\book\lib\tab. lib

WCL386 识别编译参数 %1 并且把 . OBJ 文件链接到 TAB. LIB 文件上。

使用 CCL. BAT 编译并链接 CUR1. C ，键入：

CCL　CUR1

按 Enter 键。一旦编译、链接完成，生成一个文件 CUR1. EXP 。使用 Phar Lap 的 DOS—EXTENDER 程序来运行 CUR1. EXP ，键入：

RUN386　CUR1

按 Enter 键。CUR1. EXP 就会按我们希望的去执行。

在后面函数的叙述中，不再给出各批处理文件的清单，而只使用它们的文件名称。如果需要使用 High C 或其它 32 位 C 编译器的话，只需要生成同样名称的批处理文件，由批文件采用恰当的语法去调用恰当的程序开发工具。

### 3.1.2　移动光标到指定的行列位置，函数 mvCur(...)

函数 mvCur(...)，前面已经讨论过，因为它被包含进保护方式库中，所以现在可以拿出来使用。关于函数的源程序 MVCUR. ASM(见第九章)，我们准备用 Microsoft 的 MASM 5.1 对其进行汇编。

有一个写好的批处理文件 AMASM. BAT，被用来调用 MASM 5.1 汇编源程序。AMASM. BAT 内容如下：

MASM　/ML %1

现在，若希望用 MASM 来汇编 MVCUR. ASM ，就可用 AMASM. BAT 批处理文件，键入：

AMASM　MVCUR

按 Enter 键。MVCUR 迅速被汇编好，然后就用前面提过的 WADDLIB. BAT 把目标模块 MVCUR. OBJ 加进 TAB. LIB 中。

既然 MVCUR. ASM 被汇编好，而且 MVCUR. OBJ 也被加进 TAB. LIB 中，现在就看看它能否如设想的那样工作。编译 CUR2. C (见程序 8—1—2.)，将编译生成的目标文件 CUR2. OBJ 与 TAB. LIB 文件相链接。运行生成的可执行文件 CUR2. EXP ，演示光标如何被移动到指定的行列位置上。

程序 8—1—2. 程序 CUR2. C 源代码清单

```
/////////////////////////////////////////
// CUR2. C
// Description:
// Test function gtcur(...)
/////////////////////////////////////////
```

```
#include <stdio.h>
#include <tproto.h>

void main(void);

void main()
{
int row,col;

//move cursor to row 10 ,col 10
mvCur(10,10);

//get th current cursor location
gtCur(&row,&col);

//report cursor position to screen
printf("\nRow=%02d Col=%02d\n",row,col);

}
```

### 8.1.3 相对移动光标位置,函数 rmvCur(...)

有时会遇到这种情况,需要把光标移到相对于当前位置的某个地方。起此作用的函数 rmvCur(...) 中用到了另外两个函数 gtCur(...) 和 mvCur(...) 。解决上述情况很简单:

1·取光标当前位置值

2·把目的位置相对于当前位置的偏移量值作为参数传送给 rmvCur(...),调整行列值

3·移动光标到新的位置

函数 rmvCur(...)的源代码 RMVCUR.C 见第九章。编译此源程序,把产生的目标模块 RMVCUR.OBJ 加到 TAB.LIB 中。

现在检验一下 rmvCur(...) 的功能,程序 8-1-3. 给出了演示程序 CUR3.C 。编译此程序,并把生成的 CUR3.OBJ 与 TAB.LIB 相链接。运行得到的 CUR3.EXP ,表演一下如何把光标移动到相对于当前位置的一个新的位置。

程序 8-1-3. 程序 CUR3.C 源代码清单

```
////////////////////////////////////////
//CUR3.C
//
```

```
//Description:
// Test function rmvCur(...)
//////////////////////////////////////////

// include files here
#include <stdio.h>
#include <tproto.h>

void main(void);

void main()
{
int row,col;

//move cursor to row 0 ,col 0
mvCur(0,0);

//incrementing by row and col
for (row=0,col=0;row<23;row++,col++)
{
//move cursor to row & col
mvCur(row,col);

//print message to screen startint at row ,col
printf("This line start at row %02d,column %02d",row,col);
}
}
```

### 8.1.4 保存、恢复光标位置，函数 sCloc(...)、rCloc(...)

您兴许经常需要进行一些移动光标的操作，而且希望操作完成后，光标还回到原来的地方，即保存光标－操作－恢复光标。保存光标位置，可使用 gtCur(...)先取得当前光标位置，并且将其行列值放到存储器中；而恢复光标位置可用 mvCur(...)把光标直接移到原来保存的行列位置。

当然保存、恢复光标位置，较快捷的手段还是用 sCloc(...)和 rCloc(...) 。

sCloc(...)　　取得当前光标位置并且存到静态存储器中

rCloc(...)　　从静态存储器取出光标行列值，按此值移动光标到原来位置

这是两个用起来既方便又省时的函数。

第九章给出了源程序 SCLOC. C，其中包括了 sCloc(...)和 rCloc(...)两个函数的源代码。编译 SCLOC. C，然后把 SCLOC. OBJ 放到 TAB. LIB 文件中。

程序 8—1—4. 给出的程序 CUR4. C，演示了 sCloc(...)和 rCloc(...)的使用。编译 CUR4. C，并且把生成的目标模块 CUR4. OBJ 与 TAB. LIB 相链接。运行产生的可执行文件 CUR4. EXP，演示如何对光标位置进行保存、改变、恢复。

程序 8—1—4. 程序 CUR4. C 源代码清单

```
////////////////////////////////////////////////
//CUR4.C
//
//Description:
// Test function sCloc(...)
// rCloc(...)
////////////////////////////////////////////////

//include files here
#include <stdio.h>
#include <tproto.h>

void main(void);

void main()
{
int row,col;

//save the current cursor location
sCloc();

//set cursor to page top
mvCur(5,40);

//print message to the screen at row 5 ,co 40
printf("Message printed at 5,40");

//restore proviously saved cursor location
```

```
rCloc();
}
```

8.1.5 打开、关闭光标，函数 onCur(...)、offCur(...)

有时光标是不可缺少的，它被用来标志某种位置信息，或者用来吸引人的注意力，诸如文字编辑。而在另一些场合，最好还是别让人看到它，比方一幅图画，或者屏幕弹出的一信息窗中，这时若有个闪烁的光标在屏幕上晃动，谁都会恶心。

onCur(...)　　打开光标，使可见

offCur(...)　　关闭光标，使消逝

第九章给出的程序清单 ONCUR.C，包含了 onCur(...)、offCur(...)两个函数的源代码。编译 ONCUR.C，然后把 ONCUR.OBJ 加进 TAB.LIB 中。

注意，onCur(...)和 offCur(...)调用名为 s-shape(...)、g-shape(...)的两个函数，它们系用汇编语言写成。程序清单见第九章。分别汇编 S-SHAPE.ASM 和 G-SHAPE.ASM，并且将各自的目标模块加入 TAB 库中。

程序 CUR5.C(见程序 8－1－5.)用来检测函数 onCur(...)、offCur(...)能否正常运行。编译 CUR5.C，将 CUR5.OBJ 与 TAB.LIB 链接起来。运行 CUR5.EXP 表演一下如何关闭、打开光标。

程序 8－1－5. 程序 CUR5.C 源代码清单

```
//////////////////////////////////////////////
//CUR5.C
//
//Description:
// Test function offCur(...)
// onCur(...)
//////////////////////////////////////////////

//include files here
#include <stdio.h>
#include <tproto.h>

void main()
{
int row,col;

//save the current cursor location
sCloc();
```

```
//turn the cursor off
offCur();

//set cursor to page top
mvCur(5,40);

//print message to screen at row 5,column 40
printf("Message printed at 5,40");

//wait here
getchar();

//restore proviously saved cursor location
rCloc();

//turn the cursor on
onCur();
}
```

## 8.1.6 改变光标大小,函数 sizeCur(...)

sizeCur(...)允许您用它去改变光标大小。依靠机上安装的显示卡,一个闪烁的光标一般是一个呈不同高度的矩形块。在 CGA 模式,光标高度为 8 行扫描线;EGA 模式,14 行扫描线。注意,这些数字分别和 CGA 、EGA 文本模式下的字符高度一致。

改变光标大小也很简单,把最头一条扫描线(从 0 开始)和最后一条扫描线(光标的底)传给函数 sizeCur(...) ,光标大小就此改变。

光标大小的改变可以带给屏幕显示某些变化。例如,您编写了一个文本编辑器,它有插入和改写两种编辑模式。用户处于哪种状态,您可以通过在屏幕上打印一些信息来通知用户,也可以通过调整光标大小来告诉用户。改变光标大小也能给用户一个实在的反馈,这是个较高级的方法。

函数 sizeCur(...) 使用 BIOS 改变光标大小。第九章给出了函数的源代码 SIZECUR.C ,编译 SIZECUR.C ,把生成的 SIZECUR.OBJ 加进 TAB.LIB 库中。

程序 8—1—6. 提供的程序 CUR6.C 用于检验 sizeCur(...) 的功能。编译 CUR6.C ,把编译产生的 CUR6.OBJ 与 TAB.LIB 库文件链接起来,运行 CUR6.EXP ,看看光标大小如何改变。

程序 8—1—6. 程序 CUR6.C 源代码清单

```
//////////////////////////////////////////////
//CUR6.C
//
//Description:
// Test function sizeCur(...)
//////////////////////////////////////////////

//include files here
#include <stdio.h>
#include <tproto.h>

void main(void);

void main()
{
//message 1
printf(Press any key to change cursor size.");
putchar(10);

//wait here
getchar();

//set the cursor size
sizeCur(0,12);

//message 2
printf("Press any key to restore cursor size.");
putchar(10);

//wait here
getchar();

//set cursor size
sizeCur(11,12);
}
```

### 8.1.7　保存、恢复光标大小，函数 ssizeCur(...)、rsizeCur(...)

正如前面的保存、恢复光标位置的函数一样，这里又提供了保存、恢复光标大小的两个比较快捷的函数。

ssizeCur(...)　　保存光标大小

rsizeCur(...)　　恢复原来存储的光标大小

这两个函数和 SCLOC.C 一样使用了静态存储器。

第九章中的源程序 SSIZECUR.C 包含了函数 ssizeCur(...) 和 rsizeCur(...)的源代码。编译 SSIZECUR.C，然后把 SSIZECUR.OBJ 加进 TAB.LIB 库文件中。

程序 8－1－7. 提供的 CUR7.C，用来测试 ssizeCur(...)、rsizeCur(...)的功能。编译 CUR7.C，然后把 CUR7.OBJ 和 TAB.LIB 链接起来，运行产生的可执行文件 CUR7.EXP，展示一下光标大小如何被保存、改变、恢复。

程序 8－1－7. 程序 CUR7.C 源代码清单

```
//////////////////////////////////
//CUR7.C
//
//Description:
// Test function rsizeCur(...)
// ssizeCur(...)
//////////////////////////////////

//include files here
#include <tproto.h>
#include <stdio.h>

viod main(viod);
void main()
{

//save current cursor shape
//message 1
printf("Press any key to change cursor size.");
putchar(13);

//wait here
getchar();
```

```
//set the cursor size
sizeCur(0,12);

//message 2
printf("Press any key to restore cursor size.");
putchar(13);

//wait here
getchar();

//restore previously saved cursor size
rsizeCur();
}
```

小结：

这里着重讲述了光标控制例行程序。它们对于数据录入式屏幕以及依赖于 BIOS 的视频显示例行程序都是很有用的。10 个函数提供了强有力的光标控制功能。

这一节通过对光标函数的操作，也展示了一种建库的方法。

建库的一般步骤如下：

1·生成一个库类型

2·生成一个类型函数清单

3·为函数编写源程序

4·编译或者汇编写好的源程序

5·将编译或汇编产生的目标模块加入库中

6·检测函数工作情况是否如愿

下一节将详细介绍屏幕控制例行程序。

## 第二节　屏幕控制部分

本节内容提要

· 预备知识

· 两个屏幕预定义函数

· 九个基于显示区直接存取的屏幕控制函数

· 十一个基于 BIOS 的屏幕控制函数

这节开始，先探讨有关文本方式下的光标以及屏幕控制策略。若想清楚了解使用 BIOS 或直接存储器访问写屏的信息，可翻阅其它有关资料。

保护方式下，写屏幕非常简单。然而，要是对速度和文本属性进行控制的话，写屏幕就

不是那么容易的事情了。这节分为两部分，第一部分涉及到用 BIOS 写屏幕，第二部分介绍如何使用 DOS-Extender 的局部描述表(LDT)进行屏幕存储区直接访问。

最后，相对于 BIOS，用直接存储区存取写屏幕就显得比较容易，而且具有明确的概念。实际上，文本方式下，有关屏幕的字符值、前景色、背景色、前景亮度，和前景闪烁的控制信息都包含在连续的两个字节中。第一个字节包括字符值(a,b,c,······等等)，第二个字节包括了属性值(前景色、背景色、前景亮度、前景闪烁)。

由于文本方式下屏幕是 80 个字符宽，所以计算行值很简单。彩色文本方式下屏幕存储区开始段地址定位在 0Xb800。地址 0Xb800 处，放的是字符值，地址 0Xb801 处放的是字符属性值。因为一行是 80 个字符，每个字符在存储区占两个字节，一行共占 160 个字节，所以地址 0Xb800 后再下一行字符的首地址应为 0XB8A0(0XB800＋0XA0＝0XB8A0)。表 8—2—1. 给出了一个缩简的屏幕存储区图样。

另一方面，使用 BIOS 要求您得跟踪光标位置。而且 BIOS 也不支持某些方便的屏幕控制要求。例如，您希望在屏幕上高亮度显示一串字符，就需要：

1·把屏幕上字符串读进存储区保存起来

2·按新的显示属性重写此字符串

一般来说，不管何时 BIOS 对屏幕进行操作，都显得较呆滞。

表 8—2—1. 简缩的彩色文本存储显示器

| 地　址 | 行　号 | 列　号 | 功能值 |
|---|---|---|---|
| 0XB800 | 0 | 0 | 字　符 |
| 0XB801 | 0 | 0 | 属　性 |
| 0XB802 | 0 | 1 | 字　符 |
| 0XB803 | 0 | 1 | 属　性 |
| · | · | · | · |
| · | · | · | · |
| 0XB8A0 | 1 | 0 | 字　符 |
| 0XB8A1 | 1 | 0 | 属　性 |

如果用直接存储区存取方式写屏幕，就应该为标识的屏幕字符串改属性字节。

Phar Lap 采用直接写屏方式，还有其另外的原因。Phar Lap 的 DOS--EXTENDER 自动建立一个 LDT 段选择器以直接指向屏幕。这允许您直接访问屏幕存储区。Phar Lap 的 DOS-Extender 的一个强有力的特点是，任何时候改变视频显示模式，屏幕段选择器(1ch)都自动更新。有时视频显示 RAM 区从 0XB800 开始，而有时是在 0XB000，还有的时候是在 0XA000。无论何时通过 BIOS 改变显示模式，Phar Lap 的 DOS-Extender 都自动调整 1ch 号段选择器使之直接指向屏幕 RAM 区。EGA 或者 VGA 图形程序利用 1ch 号段选择器访问图形显示。介绍这些已经超出了本书重点，但可以证实，把基于汇编的直接存储区访问的图形程序从 8086 实方式转入 80386 保护方式，和转化基于文本方式的程序比较起来并

没有更多困难。

通过直接存储区访问方式写屏幕，似乎优于 BIOS 方式，但此二种方式的例行程序都要介绍，因为不同的 DOS-Extender 使用不同的 LDT 表指向屏幕。例如，Phar Lap 的 DOS-EXTENDER 用 1CH 号段选择器指向屏幕，假如您用 1CH 号段选择器 LDT 作一写屏程序片段，而您却想让它用另一种 DOS--EXTENDER 去运行，那将显得非常荒唐。

为了使您的 .EXP 程序不至于拴死在一种 DOS-EXTENDER 上，使用 BIOS 就成了一个切实可行的方案。如果 .EXP 程序与某一种 DOS-EXTENDER 紧密结合在一起，那么就可以用直接存储区访问方式来写屏幕。

在描述具体的屏幕管理函数之前，先介绍几个术语。

- 字符　　一个 8 位值，包括了在屏幕上显示什么形状的信息
- 属性　　一个 8 位值，包含了有关字符颜色、亮度、闪烁的信息
- 标记　　一个 16 位短整型数，即在一个 8 位字符后面再附加一 8 位属性

这一节总共给出了 22 个屏幕控制函数。

2 个屏幕预定义函数：

- mkAttr(...)　　建立屏幕属性
- mkToken(...)　　建立屏幕标记

11 个基于 BIOS 的写屏幕函数：

- scrnClr(...)　　清除屏幕
- scrdChar(...)　　从屏幕读一个字符及其属性值
- scChar(...)　　往屏幕写一个字符和属性值
- scToken(...)　　往屏幕写标记
- scChr(...)　　往屏幕写字符
- scWrite(...)　　往屏幕写字符串
- scAttr(...)　　仅仅改变屏幕属性
- scVert(...)　　画一垂直文本亮条
- scHoriz(...)　　画一水平文本亮条
- scsaveScrn(...)　　保存屏幕映象
- screstScrn(...)　　恢复以前保存的屏幕

9 个基于直接存储区访问方式的写屏幕函数：

- vrdChar(...)　　从屏幕读一个字符和属性值
- vdChar(...)　　往屏幕写一个字符和属性值
- vdChr(...)　　往屏幕写一个字符
- vdWrite(...)　　往屏幕写字符串
- vdAttr(...)　　仅仅改变屏幕属性
- vdVert(...)　　画一垂直文本亮条
- vdHoriz(...)　　画一水平文本亮条
- saveScrn(...)　　保存屏幕映象
- restScrn(...)　　恢复前面保存的屏幕

## 8.2.1 屏幕预定义函数

### 8.2.1.1 **建立屏幕属性,函数** mkAttr(...)

mkAttr(...)接收用来计算8位属性值的预定义的值,它们包含在头文件TPROTO.H中。第九章提供了函数的源代码程序MKATTR.ASM,汇编之后把其目标模块加进TAB.LIB中。后面程序8—2—1.给出的程序展示了使用函数mkAttr(...)的语法。

### 8.2.1.2 **建立屏幕标记,函数** mkToken(...)

函数mkToken(...)接收两个参数,第一个为8位字符值,第二个是8位属性值。函数mkToken(...)返回值为16位屏幕标记,寄存器AX存放返回值。

第九章提供了函数的源代码MKTOKEN.ASM. 将其汇编之后把MKTOKEN.OBJ放进TAB.LIB中。使用此函数的语法及其测试程序后面再介绍。

## 8.2.2 基于BIOS的写屏幕函数

### 8.2.2.1 **清屏幕,函数** scrnClr(...)

函数scrnClr(...)使用预定义的屏幕属性清屏,并且把文本光标移到0行、0列。屏幕属性值利用函数mkAttr(...)产生。

第九章提供了函数scrnClr(...)的源程序SCRNCLR.ASM. 汇编SCRNCLR.ASM,然后把SCRNCLR.OBJ放到库文件TAB.LIB中。

程序8—2—1.提供的程序SCRN1.C演示了函数mkAttr(...)和scrnClr(...)的使用过程。编译SCRN1.C,链接SCRN1.OBJ和TAB.LIB,用Phar Lap的DOS—EXTENDER执行SCRN1.EXP,屏幕被清除,且呈现为白色前景、蓝色背景、前景亮度打开、前景闪烁关闭。

程序8—2—1. 程序SCRN1.C源代码清单

```
/////////////////////////////////////
//SCRN1.C
//
//Description:
// Test function scrnClr(...)
/////////////////////////////////////

//include files here
#include <tproto.h>

void main(void);

void main()
{
```

```
//Clear the screen with the following
//screen attribut:
//          WHITE          —> foreground
//          BLUE           —> background
//          INTENSITY      —> intensity on.
//          BLIK           —> blink off

scrnClr(mkAttr(WHITE,BLUE,ON-DENSITY,OFF-BLINK));
}
```

8.2.2.2 **从屏幕读一个字符和属性值,函数** scrdChar(...)

函数 scrdChar(...)从指明的屏幕行列位置读取屏幕标记。当此函数被调用时,光标将移到所指明的屏幕行列位置。

寄存器 AX 存放返回值,标准 C 的 16 位返回值使用此寄存器。寄存器 AH 中是屏幕属性,寄存器 AL 中是字符值。

第九章提供了函数源代码 SCRDCLR. ASM 。汇编 SCRDCLR. ASM 之后,把所得目标模块 SCRDCLR. OBJ 加入库文件 TAB. LIB 中 。

现在看一下程序 8—2—2. 中的 SCRN2. C 是如何使用函数 scrdChar(...)的。编译 SCRN2. C 之后,链接 SCRN2. OBJ 和 TAB. LIB ,运行 SCRN2. EXP ,演示如何在指明的行列位置读取字符和属性值(标记)。

程序 8—2—2. 程序 SCRN2. C 源代码清单

```
//////////////////////////////////
// SCRN2. C
//////////////////////////////////
#include <tproto. h>
void main(void);
void main()
{ char hello[12];
int col;
scrnClr(mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK));
print ("Hello Chuck!");
for (col=0;col<12;col++)
hello[col] = (char)scrdChar(0,col);
scrnClr(mkAttr(WHITE,RED,OFF INTENSITY,ON-BLINK));
puts(hell0);
}
```

8.2.2.3 **往屏幕写一个字符和属性值，函数** scChar(...)

函数scChar(...)在指明的屏幕行列位置处写一屏幕标记。第九章提供了函数的源代码清单SCCHAR.ASM，将其汇编之后把SCCHAR.OBJ 加入TAB.LIB中。

程序8－2－3. 提供的程序SCRN3.C 演示了如何使用函数scChar(...)。编译SCRN3.C，链接SCRN3.OBJ 和TAB.LIB，运行SCRN3.EXP，表演一下如何在指明的屏幕行列位置处写一个字符和属性(一个标记)。

程序8－2－3. 程序SCRN3.C 源代码清单

```
//////////////////////////////////
// SCRN3.C
//////////////////////////////////
#include <tproto.h>
void main(void);
void main()
{   char hello[] = "Hello Chuck!";
    int attr;
    int row,col;
    int token;
    int offset;
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    for (row=0,offset=0;row<20;row++,offset++)
        for(col=0;col<12;col++)
        { token = mkTiken(hello[col],attr);
        scChar(row,col + offset,token);
        }
}
```

8.2.2.4 **在当前光标位置写一个屏幕标记，函数** scToken(...)

scToken(...)在当前光标位置写一屏幕标记(一个字符和属性)。第九章给出了函数的源代码SCTOKEN.ASM，汇编之后把所得SCTOKEN.OBJ 加入TAB.LIB中。

程序8－2－4. 中的SCRN4.C，演示了使用scToken(...)的情况。编译SCRN4.C之后，链接SCRN4.OBJ 和TAB.LIB，运行SCRN4.EXP，表演一下如何往当前光标所在位置写一个字符和属性(标记)。

程序8－2－4. 程序SCRN4.C 源代码清单

```
//////////////////////////////////
// SCRN4.C
//////////////////////////////////
```

```
#include <tproto.h>
void main(void);
void main()
{   char hello[] = "Hello Chuck!";
    int attr;
    int row,col;
    int token;
    int offset;
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    for (row=0,offset=0;row<20;row++,offset++)
        for(col=0;col<12;col++)
        { token = mkTiken(hello[col],attr);
        mvCur(row,col+offset);
        scToken(token);
        }
}
```

8.2.2.5 **往屏幕光标处写字符,函数** scChr(...)

函数 scChr(...)在屏幕指定行列位置处写一个字符,存在的屏幕属性保持不变。

第九章给出了 scChr(...)的源程序 SCCHR.ASM 。汇编之后把目标模块 SCCHR.OBJ 加进 TAB.LIB 中。

编译程序 8—2—5. 中的演示程序 SCRN5.C ,链接 SCRN5.OBJ 和 TAB.LIB ,运行所得执行文件 SCRN5.EXP ,表演一下在屏幕指定行列位置处写一字符。

程序 8—2—5. 程序 SCRN5.C 源代码清单

```
////////////////////////////////////
// SCRN5.C
////////////////////////////////////
#include <tproto.h>
void main(void);
void main()
{   char hello[] = "Hello Chuck!";
    int attr;
    int row,col;
    int token;
    int offset;
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    scrnClr(attr);
```

```
    attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
    for(col=0;col<12;col++)
        scChr(0,col,hello[col]);
    for(col=0;col<12;col++)
        scChar(1,col,mkToken(hello[col],attr));
}
```

8.2.2.6 **往屏幕写字符串,函数** scWrite(...)

scWrite(...)写一指明长度的字符串,长度以字节计数,且显示遵照指明的属性。此函数可用来产生漂亮的屏幕显示。

使用 scWrite(...)的语法:

```
scWrite(row,col,length,string,attr);
```

其中 row =开始写的屏幕行,整型值

col =开始写的屏幕列,整型值

length =要写的串的字节数,整型值

string =指向字符串的指针,字符型指针

attr =要写的串的视频属性,整型值

编译 SCWRITE.C(见第九章),把 SCWRITE.OBJ 加入 TAB.LIB 中。

现在编译程序 8—2—6.中的演示程序 SCRN6.C ,链接 SCRN6.OBJ 和 TAB.LIB ,再运行 SCRN6.EXP ,表演一下如何按照指定的长度和视频属性在指定的屏幕位置写一字符串。

程序 8—2—6. 程序 SCRN6.C 源代码清单

```
/////////////////////////////////////
// SCRN6.C
/////////////////////////////////////
#include <tproto.h>
void main(void);
void main()
{   char hello[] = "Hello Chuck!";
    int attr;
    int row,col;
    int token;
    int offset;
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    scrnClr(attr);
    attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
    scWrite(0,0,12,hello,attr);
```

```
    attr = mkAttr(WHITE,GREEN,ON-INTENSITY,OFF-BLINK);
    scWrite(1,0,12,hello,attr);
}
```

8.2.2.7 **仅仅改变屏幕属性,函数** scAttr(...)

函数 scAttr(...)改变在屏幕行列位置处指定数目的显示属性。可用来生成高光亮条的菜单。

编译第九章给出的函数源代码 SCATTR.C ,把 SCATTR.OBJ 加进 TAB.LIB 中。

现在看看 SCRN7.C (见程序 8—2—7.)是如何使用 scAttr(...)的。编译 SCRN7.C ,链接 SCRN7.OBJ 和 TAB.LIB ,运行 SCRN7.EXP ,表演一下怎样改变一部分屏幕属性而不影响字符。

程序 8—2—7. 程序 SCRN7.C 源代码清单

```
////////////////////////////////////
// SCRN7.C
////////////////////////////////////
#include <tproto.h>
void main(void);
void main()
{   char hello[] = "Hello Chuck!";
    int attr;
    int row,col;
    int token;
    int offset;
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    scrnClr(attr);
    printf("%s\n",hello);
    printf("%s\n",hello);
    attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
    scAttr(0,0,12,attr);
    attr = mkAttr(WHITE,GREEN,ON-INTENSITY,OFF-BLINK);
    scAttr(1,0,12,attr);
}
```

8.2.2.8 **画一垂直亮条,函数** scVert(...)

函数 scVert(...)在指定的行列位置按指定的属性画一垂直亮条。此函数能用于将显示屏幕分成几个可视部分。

第九章给出了函数 scVert(...)的源代码 SCVERT.C,将其编译之后,把 SCVERT.

OBJ 放进 TAB. LIB 中。

现在编译程序 8—2—8. 提供的程序 SCRN8. C ，链接 SCRN8. OBJ 和 TAB. LIB ，运行所得执行文件 SCRN8. EXP ，演示一下 scVert(...)是如何使用的。

程序 8—2—8. 程序 SCRN8. C 源代码清单

```
////////////////////////////////////////////
// SCRN8. C
////////////////////////////////////////////
#include <tproto. h>
void main(void);
void main()
{   int attr;
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    scrnClr(attr);
    attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
    scVert(1,1,21,attr);
}
```

### 8.2.2.9 画一水平亮条，函数 scHoriz(...)

函数 scHoriz(...)从指定的屏幕位置开始按指定的属性画一指定字符长度的水平亮条。到后面制作屏幕矩形块和窗口时，函数 scHoriz(...)和 scVert(...)将显得非常有用。

第九章提供了函数 scHoriz(...)的源代码 SCHORIZ. C，将其编译之后，把 SCHORIZ. OBJ 加进 TAB. LIB 中。

程序 8—2—9. 中的 SCRN9. C 是使用函数 scHoriz(...) 画一水平亮条。编译 SCRN9. C ，链接 SCRN9. OBJ 和 TAB. LIB ，运行 SCRN9. EXP，看结果如何。

程序 8—2—9. 程序 SCRN9. C 源代码清单

```
/////////////////////////////////////
// SCRN9. C
/////////////////////////////////////
#include <tproto. h>
#include <stdio. h>
void main(void);
void main()
{   int attr;
    int token;
    int row;
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
```

```
    scrnClr(attr);
    attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
    scVert(1, 1,21,attr);
    scVert(1,40,21,attr);
    scHoriz(1, 1,40,attr);
    scHoriz(21,1,40,attr);
    token = mkToken(218,attr);
    scChar(1,1,token);
    token = mkToken(191,attr);
    scChar(1,40,token);
    token = mkToken(192,attr);
    scChar(21,1,token);
    token = mkToken(217,attr);
    scChar(21,40,token);
    for (row=1;row<21;row++)
        scAttr(rpw,1,40,attr);
    scWrite(1,13,14,"Chucky' s Box",attr);
    getch();
}
```

#### 8.2.2.10 保存屏幕映象，函数 scsaveScrn(...)、恢复屏幕映象，函数 screstScrn(...)

函数 scsaveScrn(...)把当前的屏幕映象放入存储器，而函数 screstScrn(...)把以前保存的屏幕映象恢复到屏幕上。当您画一窗口，而此窗口超出一部分屏幕时，这两个例行程序将显示其价值。那时您可轻易地恢复以前保存的屏幕来移动窗口。当然，如果您的程序支配某些迭盖窗口使用的话，它们不会很好地工作。不过有关窗口管理的诸多麻烦会在后面逐一解决。

第九章中的 SCSASAVE. C 给出了两个函数的源程序。编译 SCSASAVE. C，把 SC-SASAVE. OBJ 加入 TAB. LIB 中。

然后看一看程序 8—2—10. 提供的程序 SCRN10. C ，它演示了此两个函数在一弹出窗口程序中的使用。编译 SCRN10. C ，链接 SCRN10. OBJ 和 TAB. LIB ，运行 SCRN10. EXP 。它展示了一种方法，靠恢复被窗口遮盖的屏幕图像来弹出一个窗口。

程序 8—2—10. 程序 SCRN10. C 源代码清单

```
//////////////////////////////////////
// SCRN10. C
//////////////////////////////////////
#include <tproto. h>
```

```
#include <stdio.h>
void main(void);
void main()
{    int attr;
     int token;
     int row;
     sCloc();
     offCur();
     scsaveScrn();
     attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
     scrnClr(attr);
     attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
     scVert(1, 1,21,attr);
     scVert(1,40,21,attr);
     scHoriz(1, 1,40,attr);
     scHoriz(21,1,40,attr);
     token = mkToken(218,attr);
     scChar(1,1,token);
     token = mkToken(191,attr);
     scChar(1,40,token);
     token = mkToken(192,attr);
     scChar(21,1,token);
     token = mkToken(217,attr);
     scChar(21,40,token);
     for (row=1;row<21;row++)
         scAttr(rpw,1,40,attr);
     scWrite(1,13,14,"Chucky' s Box",attr);
     getch();
     screstScrn();
     onCur();
     rCloc();
}
```

千万别小看了 SCRN10.C 的性能。在这节后面您还能见到此程序的一个"直接存储区访问"版本，那时您将清楚为什么在所有不会引起祸患的情况下，"直接存储区访问"总是比 BIOS 屏幕访问更让人喜爱。

## 8.2.3 基于直接存储区访问方式的写屏幕函数

### 8.2.3.1 从屏幕读一个字符和属性值，函数 vrdChar(...)

函数 vrdChar(...)与函数 scrdChar(...)使用同样的方法。记住，函数名以 sc 开头表示利用 BIOS 的函数；而以 v[r]d 打头则意味着是基于直接存储区访问方式的写屏幕函数。

vrdChar(...)从指定的屏幕位置读一个 16 位标记。注意程序 8—2—23. 中的 vrdChar(...)是为了使用 phar Lap 的 DOS—EXTENDER 而设计的。假如您希望使用别的 DOS—EXTENDER，那您简单地用自己的 DOS—EXTENDER 的存储选择器来替换 phar Lap 的 DOS—EXENDER 的 1Ch 号存储器选择器。

第九章提供了函数的源代码 VRDCHAR. ASM，汇编 VRDCHAR. ASM 之后把所得目标模块 VRDCHAR. OBJ 加入 TAB. LIB 中。

现在看一下在程序 8—2—11. 提供的 SCRN11. C 程序中函数 vrdChar(...)是如何被使用的，编译 SCRN11. C，链接 SCRN11. OBJ 和 TAB. LIB，运行 SCRN11. EXP，演示如何从指定屏幕位置读取一个标记(字符和属性)。

程序 8—2—11. 程序 SCRN11. C 源代码清单

```
//////////////////////////////////////
// SCRN11. C
//////////////////////////////////////
#include <tproto. h>
void main(void);
void main()
{    int attr;
     char hello[12];
     int col;
     attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
     scrnClr(attr);
     printf("Hello chuck!");
     for (col=0;col<12;col++)
         hello[col] = (char)vrdChar(0,col);
     attr = mkAttr(WHITE,RED,OFF-INTENSITY,ON-BLINK);
     scrnClr(attr);
     puts(hello);
}
```

### 8.2.3.2 往屏幕写一个字符和属性值，函数 vdChar(...)

函数 vdChar(...)与函数 scChar(...)工作方法相同，只不过 vdChar(...)利用的是直

接存储器访问方式。函数 vdChar(...)允许您在指定的屏幕位置写一个 16 位屏幕标记(一字符和属性值)

第九章提供了函数 vdChar(...)源代码清单 VDCHAR.ASM，汇编 VDCHAR.ASM 然后把所得目标模块加入库文件 TAB.LIB 中。

现在编译程序 8—2—12. 给出的程序 SCRN12.C，链接 SCRN12.OBJ 和 TAB.LIB，最后运行 SCRN12.EXP，看看情况如何。函数 vdChar(...)要比 scChar(...)运行得快。

程序 8—2—12. 程序 SCRN12.C 源代码清单

```
//////////////////////////////////////
// SCRN12.C
//////////////////////////////////////
#include <tproto.h>
void main(void);
void main()
{   char hello[]="Hello chuck!";
    int attr;
    int row,col;
    int token;
    int offset;
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    for (row=0,offset=0;row<20;row++,offset++)
        for (col=0;col<12;col++)
        { token = mkToken(hello[col],attr);
          vdChar(row,col + offset,token);
        }
}
```

### 8.2.3.3 往屏幕写一个字符，函数 vdChr(...)

函数 vdChr(...)往指明的屏幕位置处写一个字符，当然是使用直接存储器访问方式。函数的源代码 VDCHR.ASM 见第九章，将其汇编之后把 VDCHR.OBJ 放进 TAB.LIB 中。

现在看看程序 8—2—13. 中的程序 SCRN13.C 是怎样运用 vdChr(...)的。汇编 SCRN13.C，然后链接 SCRN13.OBJ 和 TAB.LIB，再运行 SCRN13.EXP，演示如何在指定的屏幕位置写一字符。屏幕属性保持不变。

程序 8—2—13. 程序 SCRN13.C 源代码清单

```
//////////////////////////////////////
// SCRN13.C
```

```
////////////////////////////////////
#include <tproto.h>
void main(void);
void main()
{   char hello[]="Hello chuck!";
    int attr;
    int row,col;
    int token;
    int offset;
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    scrnClr(attr);
    attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
    for (col=0;col<12;col++)
        vdChr(0,col,hello[col];
    for (col=0;col<12;col++)
        vdChar(0,col,mkToken(hello[col],attr);
    mvCur(5,0);
}
```

8.2.3.4 **往屏幕写字符串，函数** vdWrite(...)

vdWrite(...)在屏幕指定行列位置写一指定长度和属性的字符串，使用直接存储器访问方式。使用此函数的语法与前面的函数 scWrite(...)一致。

第九章提供了函数的源程序 VDWRITE.ASM，汇编之后，把 VDWRITE.OBJ 加入 TAB.LIB 中。

现在编译程序 8－2－14. 提供的程序 SCRN14.C，链接 SCRN14.OBJ 和 TAB.LIB，最后运行 SCRN14.EXP，表演如何利用函数 vdWrite(...)在指定的屏幕位置按指定的属性写一指定长度的字符串。

程序 8－2－14. 程序 SCRN14.C 源代码清单

```
////////////////////////////////////
// SCRN14.C
////////////////////////////////////
#include <tproto.h>
void main(void);
void main()
{   char hello[]="Hello chuck!";
    int attr;
    int row,col;
```

```
    int token;
    int offset;
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    scrnClr(attr);
    attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
    vdWrite(0,0,12,hello,attr);
    attr = mkAttr(WHITE,GREEN,ON-INTENSITY,OFF-BLINK);
    vdWrite(1,0,12,hello,attr);
    mvCur(5,0);
}
```

8.2.3.5 **仅仅改变屏幕属性,函数** vdAttr(...)

函数 vdAttr(...)改变从指定屏幕位置起预先确定好长度的屏幕文本属性。它在编写数据入口和用户界面程序时非常有用。

第九章给出了函数的源代码清单 VDATTR. ASM ,将其汇编之后,把 VDATTR. OBJ 加进库文件 TAB. LIB 中。

程序 8—2—15. 中的程序 SCRN15. C 使用了 vdAttr(...)。编译 SCRN15. C ,然后链接目标文件 SCRN15. OBJ 和库文件 TAB. LIB ,再运行 SCRN15. EXP ,演示如何改变指定屏幕位置处指定数目的字符属性。

程序 8—2—15. 程序 SCRN15. C 源代码清单

```
////////////////////////////////////
// SCRN15. C
////////////////////////////////////
#include <tproto. h>
void main(void);
void main()
{   char hello[]="Hello chuck!";
    int attr;
    int row,col;
    int token;
    int offset;
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    scrnClr(attr);
    printf("%s\n",hello);
    printf("%s\n",hello);
    attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
    vdAttr(0,0,12,attr);
```

```
    attr = mkAttr(WHITE,GREEN,ON-INTENSITY,OFF-BLINK);
    vdAttr(1,0,12,attr);
}
```

8.2.3.6　**画一垂直文本亮条，函数** vdVert(...)

函数 vdVert(...)从指定的屏幕位置开始按某种属性画一条若干行高度的垂直文本亮条。一旦您需要把一显示屏幕分成有组织的几部分，那别忘了这个例行程序。

第九章给出了函数 vdVert(...)的源代码 VDVERT.C 。编译 VDVERT.C ，然后把所得目标模块 VDVERT.OBJ 加进 TAB.LIB 中。

编译程序 8－2－16.中的程序 SCRN16.C ，链接所得目标模块 SCRN16.OBJ 和库文件 TAB.LIB ，最后运行 SCRN16.EXP，演示如何使用函数 vdVert(...)在指定的屏幕位置用指定属性画一指定高度的亮条。

程序 8－2－16. 程序 SCRN16.C 源代码清单

```
/////////////////////////////
// SCRN16.C //
/////////////////////////////
#include <tproto.h>
void main(void);
void main()
{     int attr;
      attr = mkAttr(WHITE,BLUE,ON—INTENSITY,OFF—BLINK);
      scrnClr(attr);
      attr = mkAttr(WHITE,RED,ON--INTENSITY,OFF—BLINK);
      vdVert(1,1,21,attr);
      mvCur(22,0);
}
```

8.2.3.7　**画一水平文本亮条，函数** vdHoriz(...)

函数 vdHoriz(...)从指定的屏幕位置开始按某种属性画一条若干列宽度的水平文本亮条。在绘制屏幕边框的时候，您会用到函数 vdHoriz(...)和 vdVert(...)。

第九章提供了函数 vdHoriz(...)的源代码 VDHORIZ.C ，编译之后把所得目标模块 VDHORIZ.OBJ 放进库文件 TAB.LIB 中。

程序 8－2－17. 中的 SCRN17.C 使用函数 vdHoriz(...)和 vdVert(...)在屏幕上迅速弹出一个窗口。编译 SCRN17.C ，链接 SCRN17.OBJ 和 TAB.LIB，运行 SCRN17.EXP，我们可以感觉到这些函数 vd...(...)协同作战产生的力量。

程序 8—2—17. 程序 SCRN17. C 源代码清单

```
////////////////////////////////////
// SCRN17. C
////////////////////////////////////
#include <tproto. h>
#include <stdio. h>
void main(void);
void main()
{   int attr;
    int token;
    int row;
    offCur();
    scsaveScrn();
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    scrnClr(attr);
    attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
    scVert(1, 1,21,attr);
    scVert(1,40,21,attr);
    scHoriz(1, 1,40,attr);
    scHoriz(21,1,40,attr);
    token = mkToken(218,attr);
    vdChar(1,1,token);
    token = mkToken(191,attr);
    vdChar(1,40,token);
    token = mkToken(192,attr);
    vdChar(21,1,token);
    token = mkToken(217,attr);
    vdChar(21,40,token);
    for (row=1;row<21;row++)
        vdAttr(row,1,40,attr);
    vdWrite(1,13,14,"Chucky' s Box",attr);
    getch();
    onCur();
    scrnClr(7);
}
```

8.2.3.8 **保存屏幕映象,函数** saveScrn(...)、
**恢复屏幕映象,函数** restScrn(...)

函数 saveScrn(...)利用"存储器直接访问"将屏幕信息存入一内部存储缓冲区。函数 restScrn(...)把以前保存的屏幕恢复到可视屏幕上来。这两个保存、恢复屏幕的例行程序比基于 BIOS 的两个程序快得多。

第九章提供的 SCRNSAVE.C,包含了 saveScrn(...)、restScrn(...)的源代码。编译 SCRNSAVE.C 之后,把 SCRNSAVE.OBJ 加入库 TAB.LIB 中。

现在看一下程序 8—2—18. 中的程序 SCRN18.C ,它的操作将形象地向我们显示为何通过"直接存贮器存取"方法访问屏幕总是优于使用 BIOS 的方法。

程序 8—2—18. 程序 SCRN18.C 源代码清单

```
///////////////////////////////////////////
// SCRN18.C
///////////////////////////////////////////
#include <tproto.h>
#include <stdio.h>
void main(void);
void main()
{    int attr;
     int token;
     int row;
     sCloc();
     offCur();
     scsaveScrn();
     attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
     scrnClr(attr);
     attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
     scVert(1, 1,21,attr);
     scVert(1,40,21,attr);
     scHoriz(1, 1,40,attr);
     scHoriz(21,1,40,attr);
     token = mkToken(218,attr);
     scChar(1,1,token);
     token = mkToken(191,attr);
     scChar(1,40,token);
     token = mkToken(192,attr);
     scChar(21,1,token);
```

```
    token = mkToken(217,attr);
    scChar(21,40,token);
    for (row=1;row<21;row++)
        scAttr(row,1,40,attr);
    scWrite(1,13,14,"Chucky' s Box",attr);
    getch();
    screstScrn();
    saveScrn();
    printf("\n\nPress any key for Direct Video Access Demo");
    getch();
    attr = mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK);
    scrnClr(attr);
    attr = mkAttr(WHITE,RED,ON-INTENSITY,OFF-BLINK);
    scVert(1, 1,21,attr);
    scVert(1,40,21,attr);
    scHoriz(1, 1,40,attr);
    scHoriz(21,1,40,attr);
    token = mkToken(218,attr);
    vdChar(1,1,token);
    token = mkToken(191,attr);
    vdChar(1,40,token);
    token = mkToken(192,attr);
    vdChar(21,1,token);
    token = mkToken(217,attr);
    vdChar(21,40,token);
    for (row=1;row<21;row++)
        vdAttr(row,1,40,attr);
    vdWrite(1,13,14,"Chucky' s Box",attr);
    getch();
    restScrn();
    onCur();
    rCloc();
}
```

小结：

这一节提供了一个内容详实的保护方式例行程序库。以字母"sc"打头的一类程序使用 BIOS 访问屏幕，而以"vd"开头的程序利用的是直接视频存储区访问。前面演示了用直接视频存储区访问方式编制的屏幕控制程序是极具专业性的。

LDT 段选择器是由 DOS-Extender 提供的，使用 LDT 段选择器来完成直接视频存储区访问也已经作过解释。

## 第三节　发声控制部分

本节内容提要

- 发声的基本理论
- 声音开关函数，onSound(...)、offSound(...)
- 声音延时函数，Delay(...)
- 发声函数，beep(...)、bleep(...)
- 发声演示程序，SOUND3.C

计算机发出美妙的声音，必定是件令人欣喜的事情。

有些较高级的计算机，配有内部发声芯片，产生声音异常简单。靠其 MIDI(音乐设备数字接口)，编制程序来演奏乐器，其音乐效果几乎达到乱真之境界。

而 IBM PC 发声功能之可怜，令人着实伤心。因为它没有 MIMD 接口，也没有专门的发声芯片。PC 机只能从扬声器发出单调的嘟嘟声。有为 PC 机配置的发声机，但实际上似乎并未有多少人用过。但对一个程序员来说，编程控制可怜的 PC 机扬声器，虽属无奈却也是件比较实际的事情。

从另一角度来讲，我们可以在有限的环境下，寻找有创造性的办法使 PC 机产生较好的声音效果。

这里叙述了两种不同的发声方法，而且给出了两个基本的发声程序，如下

　　onSound(...)　　使扬声器按规定的频率振动

　　offSound(...)　　使扬声器停止振动

由这两个程序又衍生出另两个程序：

　　beep(...)

　　bleep(...)

在给出这两个程序之前，先在 PC 机的能力范围内探讨一下声音的概念。人们听到的声音是指中频范围的振动波，有三个可以控制的变量：

　　声音强度 — 波动的幅度

　　声音高度 — 波动的频率，每秒振动的次数

　　声音持续时间 — 起振到停止的时间

变动声音的强度、高度、持续时间，在听觉上就能造就丰富的变化。

仔细看看 PC 机的扬声器，用定时器使其产生振荡，其实是件挺简单的事情。您可以使扬声器按一定的速率振动，然后改变速率，再使其停止振动。乍一看，这种操纵声音的机制也太单调了，不过我们马上可以使其丰富起来。

调幅，操纵的是声音强度。由于 PC 机的定时器只能改变振动频率而不能改变幅度，

所以声音强度在此只有两个值，即打开、关闭。因此，在您的发声算法中，仅能有方波存在，意即声波幅度只能是 0 或者另一个固定值。

调频，主要是对频率信号进行调制。在 PC 机上，频率的变化足够使用，令人高兴的是用定时器操纵频率异常简便。

一个保真的基音称为载波。比方，一个 1000Hz 的载波，若想调制其波频率，可以使其频率不断变化，使之在 1 秒钟之内从 900Hz 升到 1100Hz，然后再返回 900Hz。我们还可增大此范围，比方在 200Hz 和 1800Hz 之间往返变化。还可以改变载波的波频，例如，100Hz、18Hz、2Hz，等等。这都是改变声音质量的重要因素。

当然，您可以改变这些因素中的一个、几个，或者全部。

程序 8－3－3. 给出了一个演示程序。它通过操纵载波频率、调频幅度、调频速率产生了一些奇妙的声音。

其实，关于发声还有很多要做而且值得去做的事情。

### 8.3.1 开、关扬声器，函数 onSound(...)、offSound(...)

函数 onSound(...)，开启定时器，来回振动 PC 机的扬声器。可以通过传递一个参数来控制振动频率。

函数 offSound(...)，停止扬声器的振动。

第九章提供了 onSound(...)的源程序 ONSOUND. ASM 以及 offSound(...)的源程序 OFFSOUND. ASM 。

分别汇编 ONSOUND. ASM 和 OFFSOUND. ASM，并且把它们各自的目标模块加入 TAB. LIB 文件中。

### 8.3.2 发声延时，函数 Delay(...)

函数 Delay(...) 接收两个控制发声延时的变量。函数 Delay(...)中的持续时间还要受 PC 机速的控制。

编译函数源代码 DELAY. C(见第九章)，然后把 DELAY. OBJ 加进 TAB. LIB 库文件中。

### 8.3.3 发声函数 beep(...)

调用函数 beep(...)，给扬声器发送一个单一的载波频率，且使之保持不变，扬声器就会发出“嘟嘟”声。第九章给出了函数的源程序 BEEP. C ，编译 BEEP. C 之后，把 BEEP. OBJ 加进库文件 TAB. LIB 中。

程序 8－3－1. 提供了表演程序 SOUND1. C ，它用到了函数 onSound(...) 、offSound(...) 、Delay(...) 、和 beep(...) 。编译 SOUND1. C ，将 SOUND1. OBJ 和 TAB. LIB 进行链接，运行 SOUND1. EXP，听听发出的“嘟嘟”声。

程序 8－3－1. 程序 SOUND1. C 源代码清单

```
////////////////////////////////////
//SOUND1. C
//
//Description:
// test: onSound(. . . )
// offSound(. . . )
// Delay(. . . )
// Beep(. . . )
////////////////////////////////////
//include files
#include <stdio. h>
# include <tproto. h>

//founction prototypes
void main(void);

void main()
{
    //constant tone beep
    beep();
}
```

## 8. 3. 4 发声函数 bleep(. . . )

函数 bleep(. . . ) 使扬声器发出不同于 beep(. . . )的另一种声音。第九章给出了函数的源代码清单 BLEEP. C ,编译 BLEEP. C 之后,把 BLEEP. OBJ 加进库文件 TAB. LIB 中。

程序 8－3－2. 提供了表演程序 SOUND2. C ,编译之后,将 SOUND2. OBJ 和 TAB. LIB 进行链接,运行 SOUND2. EXP,听听发出的什么声音。

程序 8－3－2. 程序 SOUND2. C 源代码清单

```
////////////////////////////////////
//SOUND2. C
//
//Description:
// tests:bleep(. . . )
////////////////////////////////////
```

```
//include files
#include <stdio.h>
#include <tproto.h>

//founction prototypes
void main(void);

void main()
{
    //constant tone beep
    bleep();
}
```

### 8.3.5 一个声音奇妙的表演程序

程序 8－3－3. 提供的表演程序 SOUND3.C，能够发出比较丰富的声音，它用到了前面探讨过的一些声音调制技术。请注意 SOUND3.C 中的注释，有助于了解更多产生这种声音的信息。第 8.4.1 节给出的函数 gtKey(...) 对于运行 SOUND3.C 也是必不可少的。

程序 8－3－3. 程序 SOUND3.C 源代码清单

```
//////////////////////////////////
// SOUND3.C
//////////////////////////////////
#include <stdio.h>
#include <tproto.h>
void main(void);
void snd1(void);
void snd2(void);
void snd3(void);
void snd4(void);
void snd5(void);
void snd6(void);
void snd7(void);
void snd8(void);
void snd9(void);
void snd10(void);
void snd11(void);
void snd12(void);
```

```
void snd13(void);

void main()
{       printf("Constant tone beep --> Press any key to hear \n");
        gtKey();
        bleep();
        printf("Press any key to continue \n");
        gtKey();
        printf("Sleeping bleep -->Press any key to hear \n");
        gtKey();
        bleep();
        printf("Press any key to continue \n");
        gtKey();
        printf("FM Demo 1 -->Press any key to hear \n");
        gtKey();
        snd1();
        printf("Press any key to continue \n");
        gtKey();
        printf("FM Demo 2 -->Press any key to hear \n");
        gtKey();
        snd2();
        printf("Press any key to continue \n");
        gtKey();
        printf("FM Demo 3 -->Press any key to hear \n");
        gtKey();
        snd3();
        printf("Press any key to continue \n");
        gtKey();
        printf("FM Demo 4 -->Press any key to hear \n");
        gtKey();
        snd4();
        printf("Press any key to continue \n");
        gtKey();
        printf("FM Demo 5 -->Press any key to hear \n");
        gtKey();
        snd5();
        printf("Press any key to continue \n");
        gtKey();
```

```
printf("FM Demo 6 -->Press any key to hear \n");
gtKey();
snd6();
printf("Press any key to continue \n");
gtKey();
printf("FM Demo 7 -->Press any key to hear \n");
gtKey();
snd7();
printf("Press any key to continue \n");
gtKey();
printf("FM Demo 8 -->Press any key to hear \n");
gtKey();
snd8();
printf("Press any key to continue \n");
gtKey();
printf("FM Demo 9 -->Press any key to hear \n");
gtKey();
snd9();
printf("Press any key to continue \n");
gtKey();
printf("FM Demo 10 -->Press any key to hear \n");
gtKey();
snd10();
printf("Press any key to continue \n");
gtKey();
printf("FM Demo 11 -->Press any key to hear \n");
gtKey();
snd11();
printf("Press any key to continue \n");
gtKey();
printf("FM Demo 12 -->Press any key to hear \n");
gtKey();
snd12();
printf("Press any key to continue \n");
gtKey();
printf("FM Demo 13 -->Press any key to hear \n");
gtKey();
snd13();
```

```
        printf("Press any key to continue \n");
        gtKey();
}

void snd1()
{       int cnt;
        int tone;
        tone = 100;
        for (cnt=0;cnt<200;cnt++)
        {   onSound(1000 + tone);
             Delay(100,50);
             onSound(1000 - tone);
             Delay(100,50);
        }
        offSound();
}

void snd2()
{       int cnt;
        int tone;
        tone = 700;
        for (cnt=0;cnt<200;cnt++)
        {   onSound(1000 + tone);
             Delay(100,50);
             onSound(1000 - tone);
             Delay(100,50);
        }
        offSound();
}

void snd3()
{       int cnt;
        int tone;
        tone = 100;
        for (cnt=0;cnt<50;cnt++)
        {   onSound(1000 + tone);
             Delay(400,50);
             onSound(1000 - tone);
```

```
        Delay(400,50);
    }
    offSound();
}

void snd4()
{   int cnt;
    int tone;
    tone = 700;
    for (cnt=0;cnt<50;cnt++)
    {   onSound(1000 + tone);
        Delay(400,50);
        onSound(1000 - tone);
        Delay(400,50);
    }
    offSound();
}

void snd5()
{   int cnt;
    int tone;
    int sweep;
    tone = 100;
    for (sweep=0,cnt=0;cnt<200;sweep+=20,cnt++)
    {   onSound(sweep + tone);
        Delay(100,50);
        onSound(sweep - tone);
        Delay(100,50);
    }
    offSound();
}

void snd6()
{   int cnt;
    int tone;
    int sweep;
    tone = 700;
    for (sweep=0,cnt=0;cnt<200;sweep+=20,cnt++)
```

```
	{	onSound(sweep + tone);
		Delay(100,50);
		onSound(sweep - tone);
		Delay(100,50);
	}
	offSound();
}

void snd7()
{	int cnt;
	int tone;
	int sweep;
	tone = 100;
	for (sweep=500,cnt=0;cnt<50;sweep+=80,cnt++)
	{	onSound(sweep + tone);
		Delay(400,50);
		onSound(sweep - tone);
		Delay(400,50);
	}
	offSound();
}

void snd8()
{	int cnt;
	int tone;
	int sweep;
	tone = 700;
	for (sweep=500,cnt=0;cnt<50;sweep+=80,cnt++)
	{	onSound(sweep + tone);
		Delay(400,50);
		onSound(sweep - tone);
		Delay(400,50);
	}
	offSound();
}

void snd9()
{	int cnt;
```

```
        int tone;
        int sweep;
        tone = 100;
        for (sweep=2000,cnt=0;cnt<200;sweep--=20,cnt++)
        {   onSound(sweep + tone);
            Delay(100,50);
            onSound(sweep - tone);
            Delay(100,50);
        }
        offSound();
}

void snd10()
{       int cnt;
        int tone;
        int sweep;
        tone = 700;
        for (sweep=2000,cnt=0;cnt<200;sweep--=20,cnt++)
        {   onSound(sweep + tone);
            Delay(100,50);
            onSound(sweep - tone);
            Delay(100,50);
        }
        offSound();
}

void snd11()
{       int cnt;
        int tone;
        int sweep;
        tone = 100;
        for (sweep=2000,cnt=0;cnt<50;sweep--=80,cnt++)
        {   onSound(sweep + tone);
            Delay(400,50);
            onSound(sweep - tone);
            Delay(400,50);
        }
        offSound();
```

```
}

void snd12()
{       int cnt;
        int tone;
        int sweep;
        tone = 700;
        for (sweep=2000,cnt=0;cnt<50;sweep-=80,cnt++)
        {   onSound(sweep + tone);
              Delay(400,50);
              onSound(sweep - tone);
              Delay(400,50);
        }
        offSound();
}

void snd13()
{       int cnt;
        int tone;
        int sweep;
        tone = 20;
        for (cnt=0;cnt<200;cnt++)
        {   onSound(1000 + tone);
              Delay(400,50);
              onSound(1000 - tone);
              Delay(400,50);
              tone += 10;
        }
         offSound();
}
```

小结：

当前 PC 机的发声能力都相当有限，因此用户只好用 PC 机中的定时器迫使扬声器发声。在某种程度上，靠控制声音的频率就能够产生变化比较丰富的声音效果。现在，您也可以试着编写一些自己的音乐程序。

## 第四节　键盘控制部分

本节内容提要

- 预备知识
- 取按下键的 16 位码，函数 gtKey(...)
- 取 8 位字符值，函数 gtChar(...)
- 取 8 位扫描码，函数 gtScan(...)
- 取键盘标识状态，函数 gtKBflg(...)
- 取键盘等待状态，函数 gtKBstat(...)
- 从键盘取字符串，函数 prompt(...)
- 从键盘取数据串，数据类型：32 位整数，函数 inpnum(...)

拥有一些基本的 BIOS 键盘操作函数之后，我们就可以在此基础上制作出许多强有力的读键盘程序。

键盘上的每个键用一个 16 位的值来定义，它包含了 8 位字符码和 8 位扫描码，示意如下：

16位码

| bit15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

15←—— 扫描码 ——→08　07←———字符码——→00

这里给出的键盘函数使您能很容易地访问全部 16 位键码（有些书称之为 ASCII 扫描码，注意，不是 ASCII 码），也可以访问其中的 8 位字符码，当然还可访问其 8 位的扫描码。所有 16 位键码都在 KEYBOARD.H 头文件中定义过，见第七章。

读入一个按下的键，有两种可能的程序执行时序。第一种，停止执行程序，等待一次击键动作发生。一旦被击下的键被取到，立即对它处理，然后程序按自己高兴的方式接着执行。第二种，要求程序检查是否有击下的键正在等待，但程序不停止执行。如果有键击下且正处于等待状态，把它取来，并处理产生的键码，并且按需要的方式改变程序的执行。

本节 7 个键盘操作函数将有待介绍。

### 8.4.1　取 16 位扫描码和字符码，函数 gtKey(...)

函数 gtKey(...)，停止程序执行，等待有键按下。扫描和字符码返回到 32 位整数的低 16 位。这个 16 位码的定义见头文件 KEYBOARD.H（第七章）。

第九章给出了函数 gtKey(...)的源代码 GTKEY.ASM 。汇编 GTKEY.ASM ，把

GTKEY.OBJ 加进库文件 TAB.LIB 中。

程序 8－4－1. 给出了程序 KEY1.C 的清单，编译 KEY1.C ，把 KEY1.OBJ 与 TAB.LIB 相链接，运行 KEY1.EXP ，表演一下如何用函数 gtKey(...)停止程序的运行来取得按下键的 16 位字符和扫描码。

程序 8－4－1. 程序 KEY1.C 源代码清单

```
/////////////////////////////////////////
//KEY1.C
//
//Description:
// Demonstrate the use of gtKey(...)
/////////////////////////////////////////
//include files here
#include <stdio.h>
#include <tproto.h>

//function prototypes
void main(void);

void main()
{
short int key;

//wait for key press
key = gtKey();

//print the key code 16 bit value
printf("\nThe 16 bit key value is 0x%04x\n",key);
}
```

### 8.4.2 取字符，函数 gtChar(...)

函数 gtChar(...)停止程序运行并且返回一个 8 位字符值，此值交给一个 8 位字符型变量。汇编 GTCHAR.ASM(见第九章)，然后把 GTCHAR.OBJ 加进 TAB.LIB 中。

程序 8－4－2. 给出的程序 KEY2.C ，示意了如何来使用函数 gtChar(...)。编译 KEY2.C ，把 KEY2.OBJ 与 TAB.LIB 进行链接，运行 KEY2.EXP，演示如何停止程序运行并取得一个按下的键的字符码。

程序 8—4—2. 程序 KEY2. C 源代码清单

```
//////////////////////////////////////////
//KEY2.C
//
//Description:
// Demonstrate the use of gtChar(...)
//////////////////////////////////////////
//include files here
#include <stdio.h>
#include <tproto.h>

//function prototypes
void main(void);

void main()
{
      char key;

      //wait for key press
      key = gtChar();

      //print the key code 16 bit value
      printf("\nThe key character is: %c\n",key);
}
```

### 8.4.3 取扫描码,函数 gtScan(...)

函数 gtScan(...)停止程序运行,等待有一键按下,并且把其 8 位扫描码返回到一 8 位字符型变量中。第九章提供了函数 gtScan(...)的源代码 GTSCAN. ASM,汇编 GTSCAN. ASM 之后,把 GTSCAN. OBJ 加入 TAB. LIB 文件中。

编译演示程序 KEY3. C(见程序 8—4—3. ),然后把 KEY3. OBJ 与 TAB. LIB 链接起来,再运行 KEY3. EXP ,表演一下如何使用函数 gtScan(...)停止程序运行去取得一按下键的扫描码。

程序 8—4—3. 程序 KEY3.C 源代码清单

```
////////////////////////////////////////////
//KEY3.C
//
//Description:
// Demonstrate the use of gtScan(...)
////////////////////////////////////////////
//include files here
#include <stdio.h>
#include <tproto.h>

//function prototypes
void main(void);

void main()
{
    char key;

    do
    {
        //wait for key press
        key = gtScan();

    //print the key code 16 bit value
    printf("\nThe key' s scan code is: 0x%02x\n",key);

        //keep looping until F10 pressed
    }while (key ! =(F10>>8));

    printf("\n\nF10 key pressed so return to DOS\n");
}
```

### 8.4.4 取换档键标识状态,函数 gtBKflag(...)

函数 gtBKflag(...)不等待击键动作,且返回 8 位换档键状态标识到一整数中。换档键状态标识值如下:

| 位<br>7 6 5 4 3 2 1 0 | 按下的<br>键 | 十进制<br>值 |
|---|---|---|
| 0 0 0 0 0 0 0 1 | 右 Shift | 001 |
| 0 0 0 0 0 0 1 0 | 左 Shift | 002 |
| 0 0 0 0 0 1 0 0 | Control | 004 |
| 0 0 0 0 1 0 0 0 | Alt | 008 |
| 0 0 0 1 0 0 0 0 | Scroll on | 016 |
| 0 0 1 0 0 0 0 0 | Num on | 032 |
| 0 1 0 0 0 0 0 0 | Caps on | 064 |
| 1 0 0 0 0 0 0 0 | Insert | 128 |

汇编函数的源代码 GTKBFLAG. ASM (见第九章),并且把 GTKBFLAG. OBJ 加进 TAB. LIB 中。

演示程序 KEY4. C (程序 8－4－4.)告诉我们如何使用函数 gtBKflag(...),编译 KEY4. C ,然后把 KEY4. OBJ 与 TAB. LIB 相链接,再运行 KEY4. EXP ,表演一下如何在不停止程序运行的情况下读取换档键的状态。

程序 8－4－4. 程序 KEY4. C 源代码清单

```
/////////////////////////////////
// KEY4. C
/////////////////////////////////
#include <stdio. h>
#include <tproto. h>
void main(void);
void main()
{    char flag;
     for (;;)
     {    flag = gtKBflag();
     if ((flag & RIGHT-SHIFT))
          printf("Right Shift,");
     if ((flag & LEFT-SHIFT))
          printf("Left Shift,");
     if ((flag & CTRL-PRESS))
          printf("Control,");
```

```
        if ((flag & ALT-PRESS))
            printf("Alt,");
        if ((flag & SCROLL-LOCK))
            printf("Scroll,");
        if ((flag & NUM-LOCK))
            printf("Num,");
        if ((flag & CAPS-LOCK))
            printf("Caps,");
        printf("\n");
        if ((flag & INSERT-ON))
        {   printf("Right Shift,");
             break;
        }
      }
    }
```

### 8.4.5 取击键等待状态,函数 gtKBstat(...)

函数 gtKBstat(...)不等待击键动作,并且如果键盘缓冲区中没有正在等待的键,则返回一空值(NULL)。如果有键处于等待状态,那么键的 16 位扫描和字符码作为一整型数返回。

第九章给出了函数 gtKBstat(...)的源程序清单 GTKBSTAT.ASM,汇编之后,把 GTKBSTAT.OBJ 加入库文件 TAB.LIB 中。

程序 8-4-5. 提供的程序 KEY5.C,演示了如何使用函数 gtKBstat(...)的情形。编译 KEY5.C 之后,把 KEY5.OBJ 与 TAB.LIB 进行链接,再运行 KEY5.EXP,表演一下怎样在不停止程序执行的情况下,获得所按键的 16 位扫描和字符码。

程序 8-4-5. 程序 KEY5.C 源代码清单

```
//////////////////////////////////
// KEY5.C
//////////////////////////////////
#include <stdio.h>
#include <tproto.h>
void main(void);
void main()
{   short int key;
    for (;;)
    {   key = gtKBstat();
```

```
    if (! key)
     printf("No key waiting \n");
  else
  {    printf("The key' s 16 bit code is: 0x%04x",key);
    printf("\nPress any key to continue \n");
    gtKey();
    }
  if (key == F10)
    break;
   }
}
```

### 8.4.6 从键盘取一字符串,函数 prompt(...)

函数 prompt(...),从键盘取一预定义长度的字符串,在屏幕上显示字符,并且把字符串返回到一字符缓冲器中。Backspace(←)键取消字符且左移光标。按 Esc 键终止 prompt(...)的执行,返回值为 FALSE 。按 Enter 键终止函数运行,返回值等于 TRUE 。使用函数 prompt(...)的语法为:

ret-val=prompt(string,length);

其中 ret-val 整型数,取函数返回值,如果函数被 Enter 键终止,它等于 TRUE ,而当函数被 Esc 键终止的情况下,它等于 FALSE。

string 字符型指针(char *string),等于从键盘敲入的字符,这字符要放到缓冲器中由 string 指向的位置。

length 整型值,等于键入字符串的长度。

第九章提供了函数的源程序代码 PROMPT.C ,编译,然后把 PROMPT.OBJ 放入库文件 TAB.LIB 之中。

程序 8—4—6. 所示的程序 KEY6.C ,演示了函数 prompt(...)的使用。编译 KEY6.C ,链接 KEY6.OBJ 和 TAB.LIB ,运行 KEY6.EXP ,看一下如何从键盘取一字符串,并且放到缓冲区中。

程序 8—4—6. 程序 KEY6.C 源代码清单

```
////////////////////////////////////
// KEY6.C
////////////////////////////////////
#include <stdio.h>
#include <tproto.h>
void main(void);
void main()
```

```
{   char buffer[26];
    saveScrn();
    offCur();
    sCloc();
    ssizeCur();
    scrnClr(mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK));
    sizeCur(0,14);
    printf("Test of:if(prompt(buffer,25))\n");
    printf("Enter String: ");
    vdAttr(1,14,25,mkAttr(BLACK,WHITE,OFF-INTENSITY,OFF-BLINK));
    if (prompt(buffer,25))
    {   printf("\nSize of response is %d\n",strlen(buffer));
      printf("Input response is %s\n",buffer);
    }
    else
    {   printf("ESCAPE KEY pressed — No action\n");
    }
    offcur();
    printf("\nPress key to restore screen and return to dos\n");
    gtkey();
    restScrn();
    rsizeCur();
    rCloc();
    onCur();
}


    for (;;)
    {   key = gtKBstat();
    if (! key)
         printf("No key waiting \n");
      else
      {   printf("The key' s 16 bit code is: 0x%04x",key);
         printf("\nPress any key to continue \n");
         gtKey();
      }
      if (key == F10)
         break;
    }
```

```
}
```

## 8.4.7　从键盘取十进制数，函数 inpnum(...)

函数 inpnum(...)允许用户从键盘键入一长型十进制数。在屏幕上当前光标位置显示输入的数据。按 Enter 键或者 Esc 键，函数 inpnum(...)结束。按 Enter 键结束执行，返回值为 TRUE，而按 Esc 键结束函数运行，返回值等于 FALSE。函数 inpnum(...)允许用户使用数字键"0"到"9"、Backspace、Enter、Esc 键。按下除此之外的任何键都发出嘟嘟声。

使用函数 inpnum(...)的语法：

　　ret-val=inpnum(val,length);

其中 ret-val 整数(int)，用 Enter 键中止函数运行，ret-val=TRUE。

　　而用 Esc 键中止函数执行，ret-val=FALSE。

　　val 长整型指针(long *val)，接收键入的十进制长整数。

　　length 整型值(int)，等于键入的数字串的长度。

第九章提供了函数的源程序代码 INPNUM.C，编译 INPNUM.C 之后，把所得目标模块 INPNUM.OBJ 加入库文件 TAB.LIB 中。

程序 8—4—7. 提供了函数的演示程序 KEY7.C。编译 KEY7.C，链接 KEY7.OBJ 和库文件 TAB.LIB，运行 KEY7.EXP，表演从键盘取一长型整数。

程序 8—4—7 程序 KEY7.C 源代码清单

```
/////////////////////////////////
//KEY7.C
//
//Description:
//Desonstrates the use of:
//inpnum(...)
/////////////////////////////////
//include files
#include <stdio.h>
#include <tproto.h>

//function prototypes
void main(viod);

viod main()
{
    long val;
```

```
//save the screen
saveScrn();

//turn off the cursor
offCur();

//save the cursor location
sCloc();

//save the size of the cursor
ssizeCur();

//clear the screen with designated attribute
scrnClr(mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK));

//re—size the cursor
sizeCur(0,14);

//print the message to screen
printf("Test of :if(inpnum(val,10))\n");
//print the prompt before input
printf("Enter NUMBER:");
//highlight 25 bytes where string is entered
vdAttr(1,14,25,mkAttr(BLACK,WHITE,OFF-INYENSITY,OFF-BLINK));
//call prompt for string input
if (inpnum(&val,10);
 //inpnum(...)terminated with ENTER key press
 //report string length
 printf("\nValue entered is:%d\n",val);
else
 //prompt(...)termonated with ESCAPE key press
 //report message
 printf("\nESCAPE KEY PRESSED — NO ACTION\n");
//turn the cousor off
offCur();
//print restore screen message
printf("\nPress key to restore screen and return to DOS\n");
//wait for key press
```

```
    gtKey();
    //restore original screen
    restScrn();
    //restore previously saved cursor size
    rsizeCur();
    //restor previously saved cussor location
    rCloc();
    //turn on the cursor
    onCur();
}
```

### 8.4.8 一个简单的数据入口式程序

程序 8—4—8. 提供的程序 KEY8. C ,表现了如何利用前面提到的众多函数来制作一个简的数据入口式窗口。函数 prompt(. . . )用来输入名称和地址数据。编译 KEY8. C ,链接 KEY8. OBJ 和 TAB. LIB ,运行 KEY8. EXP ,将出现一个利用键盘输入函数编制的数据输入式突出亮度的窗口。

程序 8—4—8. 程序 KEY8. C 源代码清单

```
//////////////////////////////////
// KEY8. C
//////////////////////////////////
#include <stdio. h>
#include <string. h>
#include <tproto. h>
void main(void);
void printf-card(void);
short int ad-record(void);
void verify-c-rec(void);
typedef struct {
          int number;
          char name[26];
          char street[41];
          char city[26];
          char state[3];
          char zip[6];
          char area-code[4];
          char phone[9];
```

```
}NAD;

NAD c-rec;
int rec-number=5568;

char buff32[] = {
      32,32,32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,32,32 };

void main()
{   saveScrn();
    offCur();
    sCloc();
    ssizeCur();
    scrnClr(mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK));
    sizeCur(0,14);
    print-card();
    add-record();
    verify-c-rec();
    offCur();
    gtKey();
    restScrn();
    rsizeCur();
    rCioc();
    onCur();
}

char row1[]="Client Number:                        ";
char row2[]="Client Name:                          ";
char row3[]="Street Address:                       ";
```

```
char row4[]="City:                Stat:     Zip:              ";
char row5[]="Area Code:           Phone:                      ";
void print-car()
{    int attr;

     attr=mkAttr(BLACK,WHITE,OFF-INTENSITY,OFF-BLINK);
     vdWrite(1,1,50,row1,attr);
     vdWrite(2,1,50,row2,attr);
     vdWrite(3,1,50,row3,attr);
     vdWrite(4,1,50,row4,attr);
     vdWrite(5,1,50,row5,attr);
}

static int c-cur[8][2]={1,17,2,15,3,17,4,6,4,35,4,43,5,11,5,23};
int item = 1;
short int add-record()
{    int done;
     short int key,ret-val,attr1,attr2;
     char temp[80];
     attr1=mkAttr(BLACK,WHITE,OFF-INTENSITY,OFF-BLINK);
     attr2=mkAttr(BLACK,RED,ON-INTENSITY,OFF-BLINK);
     memset(temp,' 0' ,20);
     memset(&c-rec,' 0' ,sizeof(c-rec));
     c-rec. number=++rec-number;
     sprintf(temp,"%06d",c-rec. number);
     vdWrite(c-cur[0][0],c-cur[0][1],6,temp,attr1);
     done = aFALSE;
     do
     { memset(c-rec. name,' 0' ,25);
       vdWrite(c-cur[1][0],c-cur[1][1],25,buff32,attr2);
       mvCur(c-cur[1][0],c-cur[1][1]);
     }while(! prompt(c-rec. name,25));
     vdAttr(c-cur[1][0],c-cur[1][1],25,attr1);
     done = aFALSE;
     do
     { memset(c-rec. street,' 0' ,40);
       vdWrite(c-cur[2][0],c-cur[2][1],30,buff32,attr2);
       mvCur(c-cur[2][0],c-cur[2][1]);
```

```
}while(! prompt(c-rec. street,30));
vdAttr(c-cur[2][0],c-cur[2][1],30,attr1);
done = aFALSE;
do
{ memset(c-rec. city,' 0' ,25);
  vdWrite(c-cur[3][0],c-cur[3][1],20,buff32,attr2);
  mvCur(c-cur[3][0],c-cur[3][1]);
}while(! prompt(c-rec. city,20));
vdAttr(c-cur[3][0],c-cur[3][1],20,attr1);
done = aFALSE;
do
{ memset(c-rec. state,' 0' ,2);
  vdWrite(c-cur[4][0],c-cur[4][1],2,buff32,attr2);
  mvCur(c-cur[4][0],c-cur[4][1]);
}while(! prompt(c-rec. state,2));
vdAttr(c-cur[4][0],c-cur[4][1],2,attr1);
done = aFALSE;
do
{ memset(c-rec. zip,' 0' ,5);
  vdWrite(c-cur[5][0],c-cur[5][1],5,buff32,attr2);
  mvCur(c-cur[5][0],c-cur[5][1]);
}while(! prompt(c-rec. zip,5));
vdAttr(c-cur[5][0],c-cur[5][1],5,attr1);
done = aFALSE;
do
{ memset(c-rec. area-code,' 0' ,3);
  vdWrite(c-cur[6][0],c-cur[6][1],3,buff32,attr2);
  mvCur(c-cur[6][0],c-cur[6][1]);
}while(! prompt(c-rec. area-code,3));
vdAttr(c-cur[6][0],c-cur[6][1],3,attr1);
done = aFALSE;
do
{ memset(c-rec. phone,' 0' ,8);
  vdWrite(c-cur[7][0],c-cur[7][1],8,buff32,attr2);
  mvCur(c-cur[7][0],c-cur[7][1]);
}while(! prompt(c-rec. street,8));
vdAttr(c-cur[7][0],c-cur[7][1],8,attr1);
return(aTRUE);
```

```
}

void verify-c-rec()
{    short int attr;
     attr=mkAttr(BLACK,CYAN.OFF-INTENSITY,OFF-BLINK);
     vdWrite(15,40,strlen(c-rec.name),c-rec.name,sttr);
     vdWrite(16,40,strlen(c-rec.street),c-rec.street,sttr);
     vdWrite(17,40,strlen(c-rec.city),c-rec.city,sttr);
     vdWrite(18,40,strlen(c-rec.state),c-rec.state,sttr);
     vdWrite(19,40,strlen(c-rec.zip),c-rec.zip,sttr);
     vdWrite(20,40,strlen(c-rec.area-code),c-rec.area-code,sttr);
     vdWrite(21,40,strlen(c-rec.phone),c-rec.phone,sttr);
}
```

**小结:**

**这一节提供了一些基于 BIOS 的读键盘函数。键盘上每个键与一个 16 位值相对应。这个 16 位值包含一个 8 位字符码和一个 8 位扫描码。键盘所有 16 位键码的定义在第七章的头文件 KEYBOARD.H 中给予介绍。**

**这些例行程序提供如下键盘操作方式:停止程序运行等待击键;不中止执行程序寻找击键,检查键盘标识状态。**

## 第五节　矩形块管理部分

本节内容提要

- 矩形块与窗口的区别
- 设置 RECT 结构,函数 setRect(...)
- 定义矩形屏幕区域,函数 sizeRect(...)
- 清除屏幕的矩形区域,函数 clrRect(...)
- 按属性填充矩形块,函数 fillRect(...)
- 移动 RECT 的坐标,函数 shftRect(...)
- 扩展 RECT 的坐标,函数 expdRect(...)
- 清除矩形块且画出边框,函数 boxRect(...)
- 存储矩形屏幕区域的内容,函数 saveRect(...)
- 恢复前面保存的矩形屏幕区域内容,函数 restRect(...)
- 例程

一个程序员在程序设计和编写过程中可能特别容易忘却程序用户界面的重要性,而总是喜欢想方设法使程序用最少代码获得最快运行速度。如果您也如此的话,不能说不是

一大缺陷。

对于用户来说，没有良好的界面，那是件顶顶糟糕的事情。尤其在当今商业环境下，"视觉和触觉"这些表面效果对一个软件的销售远比程序的运行重要得多。谁都喜欢一个方便的集成开发环境。与用户交换信息的一个快捷的方法就是在屏幕上开出一些矩形块，其中含有通知用户的信息。本节就介绍了有关矩形块管理的函数，这也为下节进行窗口管理提供了一个理论基础。

矩形块和窗口的区别，关键在于坐标原点。

矩形块使用全局坐标系统，其坐标原点设在屏幕左上角 0 行 0 列位置。你若想在矩形块中写一文本，必须计算矩形块从 0 行 0 列开始的偏移量。这是个看似愚蠢但用来方便的办法。

窗口使用起来亦相当简便，它使用的是局部坐标系统，把坐标原点设在窗口左上角的边缘，效果上相当于把窗口当成了屏幕中的屏幕。当然您可以利用这里提供的 10 个矩形块管理程序编出漂亮的迭盖式窗口形象，但使用下一节给出的窗口系统将会做出更好的用户界面。

### 8.5.1 设置 RECT 结构，函数 setRect(...)、sizeRect(...)

结构 RECT 的定义源于第七章的头文件 TSTRUCT.H 。

程序 8－5－0. RECT 结构

```
////////////////////////////////////////////
//RECT structure definition
////////////////////////////////////////////

typedef struct {
        //upper left row
        int ul-row;

        //upper left column
        int ul-col;

        //lower right row
        int lr-row;

        //lower right column
        int lr-col;

        //pointer to screen image
        unsigned int *image;
```

```
}RECT;
```

RECT 结构包含了一些数值，用以描述全局屏幕左上角、右下角的行列坐标值。其中还包括一个指向缓冲区的指针，此缓冲区用于包含您在矩形块底下画的屏幕图像。

函数 setRect(...)，允许你使用说明过的 RECT 结构设置变量。这些变量包括，矩形块左上角行列的屏幕位置、右下角行列的屏幕位置。

编译第九章中的源程序 SETRECT. C，然后把 SETRECT. OBJ 加进库 TAB. LIB 中。

函数 sizeRect(...)计算分配多少存储空间来存贮画好的矩形块内的屏幕图像。在使用函数 setRect(...)、sizeRect(...)之前，必须先将它们编译好，然后加入 TAB. LIB 库中。第九章提供了 setRect(...)的源代码清单 SIZERECT. C，编译之后，把 SIZERECT. OBJ 加入 TAB. LIB 中。

程序 8—5—1. 给出的程序 RECT1. C，演示了如何来使用 setRect(...)。编译 RECT1. C，链接 RECT1. OBJ 和 TAB. LIB，运行 RECT1. EXP，展示了 setRect(...) 初始化之后的 RECT 结构的内容。

程序 8—5—1. RECT1. C

```
//////////////////////////////
// RECT1.C
//////////////////////////////
#include <stdio.h>
#include <tproto.h>
void main(void);
void main()
{    RECT *R;
     R = setRect(R,3,20,16,20);
     printf("R->ul-row = %d\n",R->ul-row);
     printf("R->ul-col = %d\n",R->ul-col);
     printf("R->lr-row = %d\n",R->lr-row);
     printf("R->lr-col = %d\n",R->lr-col);
     printf("Size of image buffer = %d\n",sizeRect(R));
}
```

### 8.5.2 从屏幕上清除矩形块区域，函数 clrRect(...)

clrRect(...)清除屏幕上矩形块区域内容，此矩形块区域是用函数 setRect(...) 提前初始化了的 RECT 结构来描述的。清除此区域用的是正常属性(属性 7=白色前景、黑色背景、亮度关闭、闪烁关闭)

第九章给出了此函数的源代码 CLRRECT. C，将其编译之后把 CLRRECT. OBJ 放进

TAB. LIB 中。注意，函数 aclrRect(...)由函数 clrRect(...)来调用，而且在清除任何屏幕矩形块之前必须先把函数 aclrRect(...)加进库 TAB. LIB 中。

aclrRect(...)通过 BIOS 按指出的屏幕属性来清除矩形块区域。程序 8—5—6. 给出函数源代码 ACLRRECT. ASM，汇编之后，把 ACLRRECT. OBJ 加进 TAB. LIB 中。

现在看一下程序 8—5—2. 给出的 RECT2. C，此程序演示了函数 clrRect(...) 的使用情况。编译 RECT2. C，然后链接 RECT2. OBJ 和 TAB. LIB。运行 RECT2. EXP，使用正常属性来清除矩形屏幕区域。

程序 8—5—2. RECT2. C

```
//////////////////////////////
// RECT2.C
//////////////////////////////
#include <stdio.h>
#include <tproto.h>
void main(void);
void main()
{   RECT  *R;
    int attr;
    attr = mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK);
    R = setRect(R,3,20,12,60);
    scrnClr(attr);
    clrRect(R);
    gtKey();
    scrnClr(7);
}
```

### 8.5.3 按属性填充矩形块，函数 fillRect(...)

函数 fillRect(...)几乎与 clrRect(...)相同，只不过函数 fillRect(...)允许使用自己指定的属性来清除屏幕矩形块的屏幕内容。

第九章给出了函数 fillRect(...)的源程序 FILLRECT. C，编译 FILLRECT. C 之后，把 FILLRECT. OBJ 放进 TAB. LIB 库文件中。

编译 RECT3. C(见程序 8—5—3.)，链接 RECT3. OBJ 和 TAB. LIB，运行 RECT3. EXP，演示如何使用函数 fillRect(...)按指定的属性填充屏幕上一矩形块区域。

程序 8—5—3. RECT3.C

```
/////////////////////////////
// RECT3.C
/////////////////////////////
#include <stdio.h>
#include <tproto.h>
void main(void);
void main()
{   RECT *R;
    int attr,attr1;
    attr = mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK);
    attr1 = mkAttr(WHITE,GREEN,OFF-INTENSITY,OFF-BLINK);
    R = setRect(R,3,20,12,60);
    scrnClr(attr);
    fillRect(R,attr1);
    gtKey();
    scrnClr(7);
}
```

### 8.5.4 移动矩形块坐标,函数 shftRect(...)

函数 shftRect(...)修改一说明过的 RECT 结构中定义的行列坐标值。此函数在制作有趣的屏幕显示时特别有用。

第九章给出了 shftRect(...)的源清单 SHFTRECT.C,编译之后把 SHFTRECT.OBJ 加进库文件 TAB.LIB 中。

现在看一下程序 8—5—4. 给出的程序 RECT4.C ,它表演了函数 shftRect(...)的使用情况,编译 RECT4.C ,链接 RECT4.OBJ 和 TAB.LIB ,运行 RECT4.EXP ,我们可以看到屏幕上显示出多重覆盖的矩形块。

程序 8—5—4. RECT4.C

```
/////////////////////////////
// RECT4.C
/////////////////////////////
#include <stdio.h>
#include <tproto.h>
void main(void);
void main()
{   RECT *R;
```

```
int attr,attr1;
attr = mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK);
R = setRect(R,0,0,9,40);
scrnClr(attr);
offCur();
printf("Press any key to continue\n");
gtKey();
attr1 = mkAttr(WHITE,RED,OFF-INTENSITY,OFF-BLINK);
fillRect(R,attr1);
bleep();
attr1 = mkAttr(WHITE,GREEN,OFF-INTENSITY,OFF-BLINK);
shftRect(R,1,2);
fillRect(R,attr1);
bleep();
attr1 = mkAttr(WHITE,MAGENTA,OFF-INTENSITY,OFF-BLINK);
shftRect(R,1,2);
fillRect(R,attr1);
bleep();
attr1 = mkAttr(WHITE,BROWN,OFF-INTENSITY,OFF-BLINK);
shftRect(R,1,2);
fillRect(R,attr1);
bleep();
attr1 = mkAttr(WHITE,CYAN,OFF-INTENSITY,OFF-BLINK);
shftRect(R,1,2);
fillRect(R,attr1);
bleep();
attr1 = mkAttr(WHITE,WHITE,OFF-INTENSITY,OFF-BLINK);
shftRect(R,1,2);
fillRect(R,attr1);
bleep();
attr1 = mkAttr(WHITE,BLACK,OFF-INTENSITY,OFF-BLINK);
shftRect(R,1,2);
fillRect(R,attr1);
bleep();

gtKey();
scrnClr(7);
onCur();
```

```
}
```

8.5.5 扩展矩形块坐标，函数 expdRect(...)

函数 expdRect(...)从一个 RECT 结构中取得经过说明和初始化的值，并且扩大矩形块区域。这意味着，缩小矩形块左上角行列值，加大右下角行列值。函数 expdRect(...)可用来扩展屏幕上矩形块。扩展矩形块和窗口正是当今用户界面中的时尚。

函数的源代码 EXPDRECT.C 见第九章。编译之后把 EXPDRECT.OBJ 加进 TAB.LIB 中。

程序 RECT5.C(程序 8-5-5.)演示了函数 expdRect(...) 如何用来扩展屏幕上的矩形块。编译 RECT5.C ，链接 RECT5.OBJ 和 TAB.LIB ，运行 RECT5.EXP ，您就会看到屏幕上的矩形块被扩展的情形。

程序 8-5-5. RECT5.C

```
//////////////////////////////
// RECT5.C
//////////////////////////////
#include <stdio.h>
#include <tproto.h>
void main(void);
void breep(void);
void main()
{   RECT  *R;
    int attr,attr1,count;
    attr = mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK);
    R = setRect(R,11,36,12,44);
    scrnClr(attr);
    offCur();
    printf("Press any key to continue\n");
    gtKey();
    Wattr1 = mkAttr(WHITE,MAGENTA,OFF-INTENSITY,OFF-BLINK);
    for (count=0;count<6;coun++)
    {   expdRect(R,1,4);
         fillRect(R,attr1);
         breep();
    }
    gtKey();
    scrnClr(7);
```

```
    onCur();
}

void breep()
{   int count;
    for (count=0;count<200;count++)
    {   onSound(1500);
        Delay(20,20);
        offSound();
        Delay(20,20);
    }
    Delay(200,200);
}
```

### 8.5.6 清除矩形块内容并且画出边框,函数 boxRect(...)

函数 boxRect(...)用指定的屏幕属性和边框类型对矩形块进行清除。边框类型如下:

| 边框类型 | | 边框线型描述 | |
|---|---|---|---|
| 标号 | 值 | 顶/底 | 左/右 |
| s-s-s-s | 0 | 单 | 单 |
| s-s-d-d | 1 | 单 | 双 |
| d-d-s-s | 2 | 双 | 单 |
| d-d-d-d | 3 | 双 | 双 |

这些预定义的边框类型参见第七章的头文件 TSTRUCT.H 。

从上面看到,函数 boxRect(...)可获得 4 种对称的边框类型。当然您还能加进更多的类型,浏览一下第九章提供的函数源代码,会发现一些必要的线索。编译 BOXRECT.C ,将 BOXRECT.OBJ 加入 TAB.LIB 中。

使用函数 boxRect(...)的语法:

```
boxRect(R,s-s-s-s,attr);
```

其中 R = 指向 RECT 结构的指针,指针类型(RECT *)
s-s-s-s = 边框类型
attr = 定义的属性

程序 8—5—6. 提供的源程序 RECT6.C 使用了函数 boxRect(...)。编译 RECT6.C ,

然后链接 RECT6. OBJ 和 TAB. LIB ，运行 RECT6. EXP ，表演一下如何清除屏幕上的矩形块区域并且沿矩形四周画一边框。

程序 8—5—6. RECT6. C

```
/////////////////////////////
// RECT6. C
/////////////////////////////
#include <stdio. h>
#in lude <tproto. h>
void main(void);
void breep(void);
void main()
{   RECT  *R;
    int attr,attr1,attr2,attr3,count;
    attr = mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK);
    attr2 = mkAttr(WHITE,CYAN,OFF-INTENSITY,OFF-BLINK);
    attr3 = mkAttr(BLACK,CYAN,OFF-INTENSITY,OFF-BLINK);
    R = setRect(R,11,36,12,44);
    saveScrn();
    sCloc();
    offCur();
    attr1 = mkAttr(WHITE,MAGENTA,OFF-INTENSITY,OFF-BLINK);
    Wfor (count=0;count<6;coun ++)
    {   expdRect(R,1,4);
        fillRect(R,D-D-D-D,attr1);
        breep();
    }
    vdAttr(R—ul-row + 1,R—>ul-col +'1,54,attr2);
    vdWrite(R—ul-row + 1,R—>ul-col + 1,14,"chuck' s Window",attr2);
    gtKey();
    restScrn();
    rCloc();
    onCur();
}

void breep()
{   int count;
    for (count=0;count<200;count++)
```

```
    {   onSound(100);
        Delay(3,3);
        offSound();
        Delay(10,10);
    }
    Delay(200,200);
}
```

### 8.5.7 保存和恢复矩形块图像,函数 saveRect(...)、restRect(...)

函数 saveRect(...) 取屏幕矩形块的位置描述,并且将部分屏幕数据放入存储器。而函数 restRect(...)则用来恢复函数 saveRect(...) 保存的内容。函数 saveRect(...) 和 restRect(...)在开发迭盖式屏幕矩形块时,极有价值。

如果您希望使用迭盖式矩形块,就应该象管理堆栈一样来管理你的所有矩形块。意思是说,最后一个弹出屏幕的应该是第一个从屏幕上移走的矩形块。若不遵守"先移走,后弹出"的原则,那您的屏幕上将是一幅面目全非的现代派抽象画。为了屏幕上矩形块的移走、弹出动作有调不紊地进行,别忘记堆栈式管理。

第九章提供了上述二函数的源程序 SAVERECT.C 和 RESTRECT.C,将它们编译之后把各自的目标模块加进 TAB.LIB 中。

程序 8-5-7. 中的程序 RECT7.C,演示了如何将 5 个迭盖式矩形块弹出、移走。注意,怎样用堆栈方式保存和恢复矩形块屏幕数据。编译 RECT7.C,然后把 RECT7.OBJ 和库文件 TAB.LIB 进行链接,运行最后的执行文件 RECT7.EXP,屏幕上就会有 5 个迭盖式窗口被弹出、移走。

程序 8-5-7. RECT7.C

```
//////////////////////////////
// RECT7.C
//////////////////////////////
#include <stdio.h>
#include <tproto.h>
void main(void);
void breep(void);
void main()
{   RECT *R1, *R2, *R3, *R4, *R5;
    int attr,attr1,attr2,attr3,attr4,attr5;
    attr = mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK);
    attr2 = mkAttr(WHITE,CYAN,OFF-INTENSITY,OFF-BLINK);
    attr3 = mkAttr(BLACK,CYAN,OFF-INTENSITY,OFF-BLINK);
```

```
R1 = setRect(R1,1,4,10,40);
R2 = setRect(R2,1+3,4+5,10+3,40+5);
R3 = setRect(R3,1+6,4+10,10+6,40+10);
R4 = setRect(R4,1+9,4+15,10+9,40+15);
R5 = setRect(R5,1+12,4+20,10+12,40+20);
sCloc();
offCur();

saveRect(R1);
attr1 = mkAttr(WHITE,MAGENTA,OFF-INTENSITY,OFF-BLINK);
boxRect(R1,D-D-D-D,attr1);
breep();

saveRect(R2);
attr2 = mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK);
boxRect(R2,D-D-D-D,attr2);
breep();

saveRect(R3);
attr3 = mkAttr(WHITE,RED,OFF-INTENSITY,OFF-BLINK);
boxRect(R3,D-D-D-D,attr3);
breep();

saveRect(R4);
attr4 = mkAttr(WHITE,GREEN,OFF-INTENSITY,OFF-BLINK);
boxRect(R4,D-D-D-D,attr4);
breep();

saveRect(R5);
attr5 = mkAttr(WHITE,BROWN,OFF-INTENSITY,OFF-BLINK);
boxRect(R5,D-D-D-D,attr5);
breep();

gtKey();

restRect(R5);
breep();
restRect(R4);
```

```
    breep();
    restRect(R3);
    breep();
    restRect(R2);
    breep();
    restRect(R1);
    breep();

    rCloc();
    onCur();
}

void breep()
{   int count;
    for (count=0;count<200;count++)
    {   onSound(100);
        Delay(5,5);
        offSound();
        Delay(30,10);
    }
    Delay(200,200);
}
```

### 8.5.8 带阴影的弹出式矩形块程序

程序 RECT8.C (程序 8—5—8. ),是 RECT7.C 的一个较花哨的版本。区别在于,此处的矩形块带有阴影,光源假定在屏幕左上角。给矩形块窗口加上了阴影让人觉得挺形象,其实也很容易。

给矩形块边缘镶上阴影的算法,如下:

1. 生成准备镶边的矩形块的 RECT 结构
2. 生成阴影边框的 RECT 结构
3. 保存阴影边框的 RECT 结构
4. 画出边框 RECT
5. 改变下边缘和左边缘的属性
6. 执行例行程序
7. 恢复阴影边框 RECT 的屏幕数据

程序 8—5—8. RECT8. C

```
//////////////////////////////
// RECT8. C
//////////////////////////////
#include <stdio.h>
#include <tproto.h>
void main(void);
void breep(void);
void shadow(RECT * ,int);
void main()
{   RECT *R1, *R2, *R3, *R4, *R5;
    RECT *SR1, *SR2, *SR3, *SR4, *SR5;
    int attr,attr1,attr2,attr3,attr4,attr5;
    int sattr;
    int screen-attr;
    int count;
    attr = mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK);
    attr2 = mkAttr(WHITE,CYAN,OFF-INTENSITY,OFF-BLINK);
    attr3 = mkAttr(BLACK,CYAN,OFF-INTENSITY,OFF-BLINK);
    sattr = mkAttr(WHITE,BLACK,OFF-INTENSITY,OFF-BLINK);
    screen-attr = mkAttr(BLACK,WHITE,OFF-INTENSITY,OFF-BLINK);
    R1 = setRect(R1,1,4,10,40);
    R2 = setRect(R2,1+3,4+5,10+3,40+5);
    R3 = setRect(R3,1+6,4+10,10+6,40+10);
    R4 = setRect(R4,1+9,4+15,10+9,40+15);
    R5 = setRect(R5,1+12,4+20,10+12,40+20);
    SR1 = setRect(SR1,1,4,10+1,40+1);
    SR2 = setRect(SR2,1+3,4+5,10+3+1,40+5+1);
    SR3 = setRect(SR3,1+6,4+10,10+6+1,40+10+1);
    SR4 = setRect(SR4,1+9,4+15,10+9+1,40+15+1);
    SR5 = setRect(SR5,1+12,4+20,10+12+1,40+20+1);

    sCloc();
    offCur();

    for (count=0;count<25;count++)
        vdAttr(count,0,80,screen-attr);
```

```
saveRect(SR1);
attr1 = mkAttr(WHITE,MAGENTA,OFF-INTENSITY,OFF-BLINK);
boxRect(R1,D-D-D-D,attr1);
shadow(SR1,sattr);
breep();

saveRect(SR2);
attr2 = mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK);
boxRect(R2,D-D-D-D,attr2);
shadow(SR2,sattr);
breep();

saveRect(SR3);
attr3 = mkAttr(WHITE,RED,OFF-INTENSITY,OFF-BLINK);
boxRect(R3,D-D-D-D,attr3);
shadow(SR3,sattr);
breep();

saveRect(SR4);
attr4 = mkAttr(WHITE,GREEN,OFF-INTENSITY,OFF-BLINK);
boxRect(R4,D-D-D-D,attr4);
shadow(SR4,sattr);
breep();

saveRect(SR5);
attr5 = mkAttr(WHITE,BROWN,OFF-INTENSITY,OFF-BLINK);
boxRect(R5,D-D-D-D,attr5);
shadow(SR5,sattr);
breep();

gtKey();

restRect(SR5);
breep();
restRect(SR4);
breep();
restRect(SR3);
```

```
    breep();
    restRect(SR2);
    breep();
    restRect(SR1);
    breep();

    rCloc();
    onCur();
}

void breep()
{   int count;
    for (count=0;count<200;count++)
    {   onSound(100);
        Delay(5,5);
        offSound();
        Delay(30,10);
    }
    Delay(600,200);
}

void shadow(RECT *R,int rect sattr)
{   int count;
    vdAttr(R->lr-row-1,R->ul-col+1,R->lr-col - R->ul-col - 1,sattr);
    for (count=R->ul-row+1;count<R->lr-row;count++)
        vdAttr(count,R->lr-col-1,1,sattr);
}
```

程序 RECT8.C 中使用了函数 breep(...)作为延时，是为了能在一瞬间看到进栈的带阴影镶边的矩形块。这种带阴影镶边的矩形块的弹出和移走几乎与不带阴影镶边的矩形块一样快。

编译 RECT8.C，链接 RECT8.OBJ 和 TAB.LIB，运行 RECT8.EXP，您会发现在 80386 保护方式下用 C 语言制作带阴影镶边的矩形块是多么容易！

小结：

**本节提供了 10 个矩形块管理函数。它们用于操纵屏幕上的矩形块区域。矩形块管理程序允许初始化 RECT 结构，清除屏幕上的矩形块区域，按指定属性填充矩形块，沿矩形块四周画边框，保存和恢复矩形块范围内的屏幕数据。利用这些函数完全可以作出一个华**

美的窗口系统。下一节就讨论应用局部坐标系统的“窗口”。

## 第六节　窗口管理部分

本节内容提要

- 预备知识
- 十九个窗口管理函数
- 例程

由于窗口管理程序较其它类型函数更富协作性，本节内容的组织与上一节稍有不同。第九章中提供了19个窗口管理函数，它们的源代码被放在同一个程序模块 Wind.C 中。在一个程序模块中给出这么庞大的文件，就是为了方便您生成自己的清单。程序8—6—1.给出了 QUIT1.C 源程序清单。这个特别冗长的文献式程序演示了窗口管理程序如何被用来生成标准的弹出式窗口。研究一下程序8—6—1.提供的清单，来引用众多的非内部窗口管理函数的语法。

程序8—6—2.中的 MENU.C 是一个综合性的菜单演示程序，展现了一种方法来生成 lotus 窗口、栅格式窗口、垂直滚动亮条式窗口、平面信息窗口。MENU.C 程序旁边的注释使用说明标识“// ”而不是“/＊　＊/”。本书的一个主要目标就是方便您从8086实方式下的编程向80386保护方式编程的转换。

本节提供的19个窗口管理函数如下：

- wvdAttr(...)　给窗口屏幕设置属性字节
- wvdChar(...)　往窗口中写标记
- wvdHoriz(...)　往窗口中写水平条
- wvdVert(...)　往窗口中写垂直条
- wvdWrite(...)　往窗口中写串
- rdImg(...)　(内部函数)
- wrImg(...)　(内部函数)
- rdWind(...)　(内部函数)
- strtWind(...)　第一次显示窗口
- setAttr(...)　置窗口缺省值
- setWind(...)　置窗口尺寸
- sizeImg(...)　(内部函数)
- remvWind(...)　从屏幕上去掉窗口
- dsyWind(...)　破坏窗口结构
- setBord(...)　置窗口边框类型
- setTitle(...)　置窗口标题
- dispWind(...)　显示窗口(第二次)
- wrBox(...)　(内部函数)

• wrWind(...)　　　（内部函数）

### 8.6.1　19个窗口管理函数

程序8—6—0. 提供了WIND结构，它包含了描述窗口可见属性的数据。多数窗口管理程序访问在WIND结构中的数据。WIND结构在第七章的头文件TSTRUCT.H中列出过。

WIND结构向您展示用到了哪些窗口参数。如果您有一些其它的窗口管理函数希望加进本书列出的清单，您可以往WIND结构和代码中加入一些变量。

程序8—6—0. WIND结构清单

```
////////////////////////////////////////////////
//WIND structure
//
//Description:
// Structure which holds parameters of window.
////////////////////////////////////////////////

typedef struct {
int ul-row;int ul-col;          //upper left row,col
int lr-row;int lr-col;          //lower right row,col
unsigned int img-size;          //window img size
unsigned int far * img-ptr;     //pointer screen image
unsigned int far * img-ptr;     //pointer wind image
int box-type;                   //border selection
int attr;                       //window attribute
int visible;                    //window on
int top-offset;                 //col offest title
int top-length;                 //length title str
int show-top;                   //display title
int bot-offset;                 //col offset title
int bot-length;                 //length title str
int show-bot;                   //display title
char * t-title;                 //ptr to t title str
char * b-title;                 //ptr to b title str
}WIND;
```

第九章中的WIND.C，是一个包含了19个窗口管理函数的很长的源文件，也是一份

冗长细致的文献资料。它揭示了窗口管理系统的内涵。这里有很多对初、中级程序员来说具有启发性的程序。在此文件上多花点时间是值得的。

编译 WIND.C，将 WIND.OBJ 加进库文件 TAB.LIB 中。

使用 WIND.C 提供的程序生成窗口的概要的步骤如下：

**窗口生成过程**

第 1 步　声明窗口结构

```
WIND *QUIT;//pointer to WIND STRUCTURE
                //named QUIT
```

第 2 步　置窗口尺寸

```
//set wind so upper left corner
//is at screen row 6,and column 25
//
//set wind so lower right corner
//is at screen row 14,and column 53
//
//Return pointer to structure
QUIT = setWind(QUIT,7,24,14,53);
```

第 3 步　定义缺省屏幕属性

```
//Set Window Attr — Fore ,back,Intensity,blink
setAttr(QUIT,mkAttr(WHITE,RED,OFF-INTENSITY,OFF-BLINK));
```

第 4 步　定义窗口变框类型

```
//Set Window Border
setBord(QUIT,D-D-D-D);
```

第 5 步　置窗口边框顶端标题

```
//Set Window Border Title
setTitle(QUIT,"QUIT");
```

第 6 步　显示窗口

```
//Display window first time
strWind(QUIT);
or...
//if window has been previously
//display using function
//strtWind(...)you should
//bypass step 1 through 5
//and use function dispWind(...)
//in this fashion
dispWind(QUIT);
```

第 7 步　必要的话写窗口消息

```
//Write Window Message
//to window QUIT (say)
//at local;
//row 2
//column 0
//5 character message
//"Hello" message
//with QUIT->attr default attribute
```

第 8 步　执行窗口操作

```
.
.
.
```

第 9 步　移走窗口以及在窗口下恢复屏幕图像

```
//Remove window and display
//original screen information.
remvWind(QUIT);
```

这些步骤在程序 QUIT1.C(程序 8—6—1.)中得到全面地执行。浏览一下每个步骤，然后与程序 QUIT1.C 作一比较，这将有助于您掌握窗口的打开、关闭操作。QUIT1.C 演示很多窗口控制函数的应用情况，您会觉得让 QUIT1.C 中的窗口发生丰富的变化也很容易。

编译 QUIT1.C，链接 QUIT1.OBJ 和 TAB.LIB，运行 QUIT1.EXP，一个在 80386 保护方式下用 C 语言编制弹出窗口的方法，将豁然出现。

程序 8—6—1. QUIT1.C

```
//////////////////////////////
// QUIT1.C
//////////////////////////////
#include <stdio.h>
#include <tproto.h>
void inform1(void);
int quit1(void);
WIND *QUIT;
char q-dat1[28] = " QUIT TSR SHELL ";
unsigned char q-dat2[30] = {
   199,196,196,196,196,196,196,196,196,196,
   196,196,196,196,196,196,196,196,196,196,
   196,196,196,196,196,196,196,196,196,182 };
char q-dat3[28] = " Quitting TSR's SHELL ";
```

```
char q-dat4[28] = " program will clear the ";
char q-dat5[28] = " screen and return you to ";
char q-dat6[28] = " the standard DOS Command ";
char q-dat7[28] = " Line environment .  ";
char q-dat8[28] = " Press (Y)es to QUIT .  ";

int quit1()
{   int key;
    QUIT = setWind(QUIT,7,24,7+10,24+29);
    setAttr(QUIT,mkAttr(WHITE,RED,OFF-INTENSITY,OFF-BLINK));
    setBord(QUIT,D-D-D-D);
    setTitle(QUIT," QUIT ");
    strtWind(QUIT);
    dispWind(QUIT);
    wvdWrite(QUIT,1,1,28,q-dat1,mkAttr(RED,WHITE,OFF-INTENSITY,
                                OFF-BLINK));
    wvdWrite(QUIT,2,0,30,q-dat2,QUIT->attr);
    wvdWrite(QUIT,3,1,28,q-dat3,QUIT->attr);
    wvdWrite(QUIT,4,1,28,q-dat4,QUIT->attr);
    wvdWrite(QUIT,5,1,28,q-dat5,QUIT->attr);
    wvdWrite(QUIT,6,1,28,q-dat6,QUIT->attr);
    wvdWrite(QUIT,7,1,28,q-dat7,QUIT->attr);
    wvdWrite(QUIT,8,0,30,q-dat8,QUIT->attr);
    wvdWrite(QUIT,9,1,28,q-dat9,QUIT->attr);
    wvdAttr(QUIT,9,12,1,mkAttr(WHITE,RED,ON-INTENSITY,ON-BLINK));
    key = gtKey();
    remvWind(QUIT);
    key &= 0x00ff;
    if ((key=='Y'||(key=='y'))
        return(aTRUE);
    else
        return(aFALSE);
}

void main()
{   if (quit1())
      printf("\nThe 'Y' or 'y' key has been pressed\n);
    else
```

```
    printf("\nThe ' Y' or ' y' key has not been pressed\n);
}
```

### 8.6.2 用户界面程序

一个详细备致的源文件 MENU.C(程序 8—6—2.)清晰地展现了如何开发带有各种窗口的用户界面,带阴影的 lotus 窗口、带阴影的栅型窗口、带阴影的滚条式窗口,和带阴影的弹出信息式窗口。把众多函数放到代码中用以生成自己的窗口界面系统也就成了顺理成章的事。在 MENU.C 这个程序上多用些时间,对减少出错是很有裨益的。

程序 8—6—2. MENU.C

```
/////////////////////////////
// MENU.c
/////////////////////////////
#include <stdio.h>
#include <tproto.h>
int tgrid(void);
void info1(void);
int tlotus(void);
int main(void);
void shadWind(RECT *,int);

int lotus-flg=0;
int lotus-item=0;
int grid-item=0;
int grid-flg=0;
int sattr;
int lattr;

WIND *FIRST;
WIND *GRID;
WIND *INFORM;
WIND *LOTUS;

RECT *RFIRST;
RECT *RGRID;
RECT *RINFORM;
RECT *RLOTUS;
```

```
char title[29] = "386 Protected Mode Program";
unsigned char i-bar[31] = {
    195,196,196,196,196,196,196,196,196,196,196,
    196,196,196,196,196,196,196,196,196,196,196,
    196,196,196,196,196,196,196,196,180 };
char item1[29] = "lotus style Menu ";
char item2[29] = "Grid Style Menu ";
char item3[29] = "Some Historical Information ";
char item5[29] = "Quit C—erious Demo ";

char menu1[47] = "Mean Mode Median Range Standard Deviation";
char mess1[47] = "Mean is the Average score of the distribution";
char mess2[47] = "Mode is the most frequent score ";
char mess3[47] = "Median is the middle score of sample ";
char mess4[47] = "Range is the distance from highest to lowest ";
char mess5[47] = "Standard ddev. is avg. distance from mean ";

int lot-map[5][2] = {1,6,  7,6,  13,8,  21,7,  28,20 };

char gmenu[21] = " SELECT A NUMBER    ";
char grid1[21]  = "        1  2  3     ";
char grid2[21]  = "        4  5  6     ";
char grid3[21]  = "        7  8  9     ";
char grid4[21]  = " Press ENTER to Exit ";

int grid-map[9][2] = {3,7,3,10,3,13,4,7,4,10,4,13,5,7,5,10,5,13};

char speed1[28] = "TSR ' C' ERIOUS History ";
unsigned char speed2[30] = {
      199,196,196,196,196,196,196,196,196,196,
      196,196,196,196,196,196,196,196,196,196,
      196,196,196,196,196,196,196,196,196,182 };
char speed3[28] = "      TSR SYSTEMS LIMITED      ";
char speed4[28] = " ———————————————— ";
char speed5[28] = " ' C' erious by len Dorfman       ";
char speed6[28] = "           &. Jay Gould          ";
char speed7[28] = " ———————————————— ";
```

```
char speed8[28] = "          Press any key to Exit.          ";

int xinverse;
int hl-tense;

int tlotus()
{   int key; int exit; int exp-a;
    exp-a = mkAttr(MAGENTA,BLUE,ON-INTENSITY,OFF-BLINK);
    if (! lotus-flag)
    {   lotus-flag = 1;
        LOTUS = setWind(LOTUS,6,20,9,68);
        RLOTUS = setRect(RLOTUS,6,20,9+1,68+1);
        saveRect(RLOTUS);
        setAttr(LOTUS,mkAttr(WHITE,BLUE,ON-INTENSITY,OFF-BLINK));
        setBord(LOTUS,S-S-S-S);
        setTitle(LOTUS,"lotus Style Window");
        strtWind(LOTUS);
    }
    else
    dispWind(LOTUS);
    shadWind(RLOTUS,lattr);
    exit = aFALSE;
    do
    {   wvdWrite(LOTUS,1,1,47,menu1,LOTUS->attr);
        wvdAttr(LOTUS,1,lot-map[lotus-item][0],
                    lot-map[lotus-item][1],
                    hl-tense);
        switah (lotus-item)
        {   case 0: wvdWrite(LOTUS,2,1,47,mess1,exp-a);
              break;
            case 1: wvdWrite(LOTUS,2,1,47,mess2,exp-a);
              break;
            case 2: wvdWrite(LOTUS,2,1,47,mess3,exp-a);
              break;
            case 3: wvdWrite(LOTUS,2,1,47,mess4,exp-a);
              break;
            case 4: wvdWrite(LOTUS,2,1,47,mess5,exp-a);
              break;
```

```
        }
        key = gtKey();
        switch(key)
    {   case RIGHT-ARROW: if (lotus-item == 4)
                    lotus-item = 0;
                    else
                    lotus-item++;
                    break;
         case LEFT-ARROW: if (lotus-item == 0)
                    lotus-item = 4;
                    else
                      lotus-item--;
                      break;
         case ENTER:      exit = aTRUE;
                    break;
        }
    }while(! exit);

    remvWind(LOTUS);
    return(lotus-item);
}

int tgrid()
{   int key; int exit;
    if (! grid-flag)
    {  grid-flag = 1;
         GRID = setWind(GRID,10,10,18,32);
        RGRID = setWind(RGRID,10,10,18+1,32+1);
        saveRect(RGRID);
        setAttr(GRID,mkAttr(WHITE,RED,OFF-INTENSITY,OFF-BLINK));
        setBord(GRID,D-D-D-D);
        setTitle(GRID,"Grid Style Winddow ");
        strtWind(GRID);
    }
    else
        dispWind(GRID);
     shadWind(RGRID,lattr);
    wvdWrite(GRID,1,1,21,gmenu,xinverse);
```

```
wvdWrite(GRID,7,1,21,grid4,GRID-->attr);
wvdWrite(GRID,7,8,5 ,"ENTER",mkAttr(WHITE,RED,
                                OFF-INTENSITY,ON BLINK));
exit = aFALSE;
do
{ wvdWrite(GRID,3,1,21,grid1,GRID-->attr);
  wvdWrite(GRID,4,1,21,grid2,GRID-->attr);
  wvdWrite(GRID,5,1,21,grid3,GRID-->attr);
  wvdAttr(GRID,grid-map[grid-item][0],
               grid-map[grid-item][1],
               3,
               xinvers);
  key = gtKey();
  switch(key)
  {  case RIGHT-ARROW:
          if (grid-item==0)||(grid-item==1)||
             (grid-item==3)||(grid-item==4)||
             (grid-item==6)||(grid-item==7))
             grid-item++;
          else
             if (grid-item==2)
                grid-item=0;
               else
                 if (grid-item==5)
                     grid-item=3;
                 else
                        grid-item=6;
           break;
     case LEFT-ARROW:
           if (grid-item==2)||(grid-item==1)||
              (grid-item==5)||(grid-item==4)||
              (grid-item==8)||(grid-item==7))
              grid-item--;
          else
              if (grid-item==0)
                grid-item=2;
                else
                  if (grid-item==3)
```

```
                        grid-item=5;
                    else
                        grid-item=8;
                break;
            case DOWN-ARROW:
                if (grid-item<=5)
                    grid-item += 3;
                else
                    if (grid-item==6)
                        grid-item=0;
                    else
                        if (grid-item==7)
                            grid-item=1;
                        else
                            grid-item=2;
                break;
            case UP-ARROW:
                if (grid-item>=3)
                    grid-item -= 3;
                else
                if (grid-item==0)
                    grid-item=6;
                    else
                        if (grid-item==1)
                            grid-item=7;
                        else
                            grid-item=8;
                break;
            case ENTER:
                exit=aTRUE;
                break;
        }
    }while(! exit);
    remvWind(GRID);
    restRect(RGRID);
    return(grid-item);
}
```

```
int info1-flag = 0;
void info1()
{   int key;
    if (! info1-flag)
    {   info1-flag=1;
        INFORM = setWind(INFORM,12-5,20-5,22-5,49-5);
        RINFORM = setWind(RINFORM,12-5,20-5,22-5+1,49-5+1);
        saveRect(RINFORM);
        setAttr(INFORM,mkAttr(BLACK,CYAN,OFF-INTENSITY,OFF-BLINK));
        setBord(INFORM,D-D-D-D);
        setTitle(INFORM,"Esoteric Information");
        strWind(INFORM);
     }
     else
            dispWind(INFORM);
    shadWind(RINFORM,lattr);
    wvdWrite(INFORM,1,1,28,speed1,
            mkAttr(CYAN,BLACK,OFF-INTENSITY,OFF-BLINK));
    wvdWrite(INFORM,2,1,28,speed2,INFORM->attr);
    wvdWrite(INFORM,3,1,28,speed3,INFORM->attr);
    wvdWrite(INFORM,4,1,28,speed4,INFORM->attr);
    wvdWrite(INFORM,5,1,28,speed5,INFORM->attr);
    wvdWrite(INFORM,6,1,28,speed6,INFORM->attr);
    wvdWrite(INFORM,7,1,28,speed7,INFORM->attr);
    wvdWrite(INFORM,8,0,30,speed2,INFORM->attr);
    wvdWrite(INFORM,9,1,28,speed8,INFORM->attr);

    key = gtKey();
    remvWind(INFORM);
    restRect(RINFORM);
}

int filter-key(int);
int filter-key(int key)
{   int row;
    row = 0;
    key &= 0x000000ff;
    if((key=='l')||(key=='L'))
```

```
        row=3;
    else if((key=='g')||(key=='G'))
            row=4;
      else if((key=='s')||(key=='S'))
              row=5;
        else if((key=='q')||(key=='Q'))
              { row=7;
                bleep();
              }

                else
              { bleep();
                bleep();
                bleep();
                bleep();
                bleep();
              }

    return row;
}

int main()
{   int key;
    int exit;
    int old-row;
    int row;
    int intense;
    int ret-val;
    int screen-attr;
    int count;
    lattr=mkAttr(WHITE,BLACK,OFF-INTENSITY,OFF-BLINK);
    sattr=mkAttr(WHITE,BLACK,OFF-INTENSITY,OFF-BLINK);
    inverse=mkAttr(BLACK,WHITE,OFF-INTENSITY,OFF-BLINK);
    hi-tense=mkAttr(WHITE,WHITE,ON-INTENSITY,OFF-BLINK);
    intense=mkAttr(WHITE,MAGENTA,ON-INTENSITY,OFF-BLINK);
    SCREEN-attr=mkAttr(BLACK,WHITE,OFF-INTENSITY,OFF-BLINK);
    for (count=0;count<25;count++)
        vidAttr(count,0,80,screen-attr);
    offCur();
    clock();
```

```
FIRST = setWind(FIRST,2,4,10,34);
RFIRST = setWind(RFIRST,2,4,10+1,34+1);
saveRect(RFIRST);
setAttr(FIRST,mkAttr(WHITE,MAGENTA,OFF-INTENSITY,OFF-BLINK));
setBord(FIRST,D-D-S-S);
setTitle(FIRST," C' erious 386 (c)1990");
strtWind(FIRST);
shadWind(RFIRST,sattr);
wvdWrite(FIRST,1,1,29,title,xinverse);
wvdWrite(FIRST,2,0,31,i-bar,FIRST->attr);
wvdWrite(FIRST,3,1,29,item1,FIRST->attr);
wvdWrite(FIRST,4,1,29,item2,FIRST->attr);
wvdWrite(FIRST,5,1,29,item3,FIRST->attr);
wvdWrite(FIRST,6,0,31,i-bar,FIRST->attr);
wvdWrite(FIRST,7,1,29,item5,FIRST->attr);
wvdAttr(FIRST,3,2,1,intense);
wvdAttr(FIRST,4,2,1,intense);
wvdAttr(FIRST,5,2,1,intense);
wvdAttr(FIRST,7,2,1,intense);
row=3;
old-row=3;
exit=aFALSE;
do
{   wvdAttr(FIRST,old-row,1,29,FIRST->attr);
      wvdAttr(FIRST,old-row,2,1,intense);
      wvdAttr(FIRST,row,1,29,xinverse);
      wvdAttr(FIRST,row,2,1,hl-tense);
      old-row=row;
      key=gtKey();
      switch(key)
      {   case DOWN-ARROW:
               if (row==7)
                    row=3;
            else if (row==5)
                    row=7;
                 else
                      row++;
            break;
```

```
            case UP-ARROW:
                    if (row==7)
                        row=5;
                else if (row==3)
                            row=7;
                        else
                            row--;
                break;
            case ENTER:
                    switch(row)
                    { case 3: tlotus(); break;
                      case 4: tgrid(); break;
                      case 5: info1(); break;
                      case 7: exit=aTRUE; break;
                    }
                    break;
            default:
                    ret-val=filter-key(key);
                    if (! ret-val)
                        break;
                    else
                        row=ret-val;
                    break;
            }
    }while(! exit);

    remvWind(FIRST);
    restRect(RFIRST);
    for(count=0;count<25;count++)
        vdAttr(count,0,80,7);
    onCur();
    rCloc();
    return(0);
}

void shadWind(RECT *R,int sattr)
{   int count;
    vdAttr(R->lr-row,R->ul-col+1,R->lr-col - R->ul-col -1,sattr);
```

```
for (count=R->ul-row+1;count<R->lr-row+1;count++)
        vdAttr(count,R->lr-col,1,sattr);
}
```

小结:

本节介绍了 19 个窗口管理函数。它们方便了对您程序中用户界面部分的开发,这一套函数允许您使用窗口局部坐标系统,挑选一种窗口边框类型,缺省屏幕属性,边框标题,以及向窗口中写入文本。窗口下的屏幕数据在窗口出现时被自动保存,在窗口被移走的情况下能被自动恢复。在您的程序中加进"窗口",这亦是一个当今程序员编程修养的一种体现。

## 第七节 鼠标管理部分

本节内容提要

- 初始化鼠标,函数 msinit(...)
- 打开鼠标,函数 mson(...)
- 关闭鼠标,函数 msoff(...)
- 取鼠标状态,函数 msstat(...)
- 例程

本节介绍如何编写一种程序,它能在文本方式下把鼠标作为一种输入设备。有关鼠标以及使用鼠标的信息,可以从鼠标驱动程序获得。访问鼠标驱动器,可通过 BIOS 的中断 33(int 33H)。

本节提供了 4 个鼠标驱动程序:

- msinit(...)　　初始化鼠标
- mson(...)　　打开鼠标
- msoff(...)　　关闭鼠标
- msstat(...)　　取鼠标状态

其中每个函数都使用了中断 33H(int 33H)。还有很多别的有关鼠标的函数也使用了中断 33H,它们不属本书讨论之列。

象前面一样,本节也提供了众多的演示程序。MOUSE3. C(程序 8—7—3)使用了 msinit(...)、mson(...)、msoff(...),和 msstat(...)四个函数,其功能为显示一鼠标控制的文本光标、让光标随鼠标的运动在屏幕上协调地移动、报告鼠标当前位置、有否鼠标按钮被按下。实际上程序 MOUSE3. C 提供了制作文本方式鼠标用户界面的必要的基本程序。

### 8.7.1 初始化鼠标,函数 msinit(...)

msinit(...)检查计算机是否安装了鼠标。若未安装,函数 msinit(...)返回值等于 0XFFFF;若安装了鼠标,函数 msinit(...)返回值为所装鼠标含有按钮的个数。msinit(...)

不接收任何参数。

函数的源代码 MSINIT. ASM 见第九章。汇编之后，将 MSINIT. OBJ 放进 TAB. LIB 中。

程序 MOUSE1. C（程序 8—7—1）使用了函数 msinit(...)。编译 MOUSE1. C，链接 MOUSE1. OBJ 和 TAB. LIB，运行 MOUSE1. EXP 来初始化鼠标。

程序 8—7—1. MOUSE1. C

```
/////////////////////////////
// MOUSE1. C
/////////////////////////////
#include <stdio. h>
#include <tproto. h>
void main(void);
void main()
{ int ret-val;
  ret-val = msinit();
  if (ret-val == oxffff)
       printf("\nThere is not mouse available. \n");
     else
       printf("\nThere are %d mouse buttons available. \n",ret-val);}
```

## 8.7.2 打开和关闭鼠标，函数 mson(...)、msoff(...)

mson(...)　　显示文本模式鼠标光标。

msoff(...)　　消除屏幕上的鼠标光标。

这两个函数没有任何返回值，也不传递任何参数。

第九章中的 MSON. ASM 和 MSOFF. ASM 是上述两个函数的源代码。将其分别汇编，并把各自的目标模块 MSON. OBJ 和 MSOFF. OBJ 分别加进 TAB. LIB 库文件中。

程序 MOUSE2. C（程序 8—7—2）使用了上述两个函数 mson(...)和 msoff(...)，编译 MOUSE. C，链接 MOUSE. OBJ 和 TAB. LIB。运行 MOUSE2. EXP，注意如何使文本方式鼠标光标打开和关闭。

程序 8—7—2. MOUSE2. C 结构

```
/////////////////////////////
// MOUSE2. C
/////////////////////////////
include <stdio. h>
include <tproto. h>
void main(void);
```

```
void main()
{ int ret-val;
  scrnClr(7);
  ret-val = msinit();
  if (ret-val == oxffff)
       printf("\nThere is not mouse available. \n");
  else
       printf("\nThere are %d mouse buttons available. \n",ret-val);
  mson();
  gtKey();
  msoff();}
```

### 8.7.3 取鼠标状态.函数 msstat(...)

函数 msstat(...)报告两个信息，一个，文本模式鼠标光标的行和列的屏幕像素位置;另一个，鼠标一些按钮被按下的状态。

使用函数 msstat(...)的语法:

batton = msstat(&X-pos,&Y-pos);

其中 button = 是否有一个鼠标安扭被按下,返回一整型值:

1 按下左按钮

2 按下右按钮

3 按下中间按钮

X-pos = 文本鼠标光标在屏上像素的 X 坐标值(整型值)

Y-pos = 文本鼠标光标在屏上像素的 Y 坐标值(整型值)

此函数不中止程序运行，它和键盘控制函数 gtKBstat(...)相仿。

先汇编函数的源代码 MSSTAT. ASM(见第九章)，然后将 MSSTAT. OBJ 加入 TAB. LIB 中。

程序 MOUSE3. C(程序 8—7—3)注意了如何在屏幕上移动文本光标，并且保存其 X 和 Y 坐标轨迹以及鼠标按钮的状态。编译 MOUSE3. C，链接 MOUSE3. OBJ 和 TAB. LIB 文件。运行 MOUSE3. EXP,您将了解如何作到下列事情:

1. 用鼠标移动文本鼠标光标
2. 报告鼠标光标像素位置值
3. 报告鼠标按钮状态

程序 8—7—3. MOUSE3. C

```
////////////////////////////////
// MOUSE3. C
////////////////////////////////
```

```
#include <stdio.h>
#include <tproto.h>
void main(void);
void main()
{ int ret-val;
  int XPOS,YPOS;
  scrnClr(mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK));
  offCur();
  ret-val = msinit();
  if (ret-val == oxffff)
       printf("\nThere is not mouse available.\n");
  else
       printf("\nThere are %d mouse buttons available.\n",ret-val);
  mson();
  while (! gtKBstat())
  { ret-val = msstat(&XPOS,&YPOS);
    mvCur(20,0);
    printf("Mouse X Posotion is %03d and Y Position is %03d.\n",XPOS,YPOS);
    printf("Mouse button value is %02d",ret-val);
    mvCur(22,0);
    if (ret-val == 1)
      printf("left Button Down");
      else
        if (ret-val == 2)
          printf("Right Button Down");
          else
               if (ret-val == 4)
                    printf("Center Button Down");
                else
                    printf("                          ");
  }
  msoff();
  onCur();
  scrnClr(7);
}
```

### 8.7.1 一个简单的动作序列处理程序

这里提供一个简单的动作序列处理程序 MOUSE4.C(程序 8—7—4)，它被用于监控鼠标和键盘的动作。这个序列展现了一个开发简单的动作序列处理程序的方法，如下：

1. 不停止程序的执行
2. 报告是否已经有键按下
3. 报告是否已经有过鼠标运动
4. 报告是否已经有过鼠标按钮被按下

动作序列处理程序的用处，在编制既对键盘输入敏感又对鼠标输入敏感的用户界面时表现得很突出。

程序 8—7—4. MOUSE4.C

```
///////////////////////////////
// MOUSE4.C
///////////////////////////////
#include <stdio.h>
#include <tproto.h>
void main(void);
void main()
{ int ret-val;
  int XPOS,YPOS;
  int key;
  scrnClr(mkAttr(WHITE,BLUE,OFF-INTENSITY,OFF-BLINK));
  offCur();
  ret-val = msinit();
  if (ret-val == oxffff)
  { scrnClr(7);
    printf("\nThere is not mouse available. \n");
  exit(0);
  }
  else
       printf("\nThere are %d mouse buttons available. \n",ret-val);
  mson();
  do
  { key = gtKBstat();
    ret-val = msstat(&XPOS,&YPOS);
    mvCur(20,0);
    printf("Mouse X Posotion is %03d and Y Position is%03d. \n",XPOS,YPOS);
```

```
    printf("Mouse button value is %02d",ret-val);
    mvCur(22,0);
    if (ret-val == 1)
    printf("left Button Down");
      else
        if (ret-val == 2)
          printf("Right Button Down");
          else
            if (ret-val == 4)
                printf("Center Button Down");
              else
                  printf("                         ");
      if (key)
      { mvCur(19,0);
        printf("Code of key pressed:0x%04X",key);
        bleep();
        bleep();
        bleep();
        mvCur(19,0);
        printf("                                    ");
      }
  }while(key! =F10);
  msoff();
  onCur();
  scrnClr(7);
}
```

小结：

这一节给出了四个鼠标操纵例行程序。函数 msinit(...)初使化鼠标，当然前提是机器上安装了鼠标，而当不存在鼠标的情况下就报告未接鼠标。函数 mson(...)打开文本鼠标光标。函数 msoff(...)关闭文本鼠标光标。函数 msstat(...)报告当前鼠标所在的行列位置，并且报告是否有鼠标按钮被按下。

# 第九章　库函数源代码清单

本章内容提要

- 按所属章节排序索引
- 按名称字典排序索引
- 源代码清单

## 第一节　按所属章节排序索引


| | | | |
|---|---|---|---|
| 8.1. | gtCur(...) | 取当前光标位置 | 176 |
| 8.1. | mvCur(...) | 移动光标到指定行列 | 176 |
| 8.1. | rmvCur(...) | 移动光标到相对于当前的某个位置 | 177 |
| 8.1. | sCloc(...) | 保存光标当前位置 | 178 |
| 8.1. | rCloc(...) | 恢复以前保存的光标位置 | 178 |
| 8.1. | offCur(...) | 关闭光标 | 178 |
| 8.1. | onCur(...) | 打开光标 | 178 |
| 8.1. | sizeCur(...) | 设置光标大小 | 180 |
| 8.1. | ssizeCur(...) | 保存光标当前大小 | 181 |
| 8.1. | rsizeCur(...) | 恢复以前保存的光标大小 | 181 |
| 8.2.1. | mkAttr(...) | 建立屏幕属性 | 182 |
| 8.2.1. | mkToken(...) | 建立屏幕标记 | 183 |
| 8.2.2. | scrnClr(...) | 清除屏幕 | 184 |
| 8.2.2. | scrdChar(...) | 从屏幕读一个字符及其属性值 | 184 |
| 8.2.2. | scChar(...) | 往屏幕写一个字符和属性值 | 185 |
| 8.2.2. | scToken(...) | 往屏幕写标记 | 186 |
| 8.2.2. | scChr(...) | 往屏幕写字符 | 187 |
| 8.2.2. | scWrite(...) | 往屏幕写字符串 | 200 |
| 8.2.2. | scAttr(...) | 仅仅改变屏幕属性 | 187 |
| 8.2.2. | scVert(...) | 画一垂直文本亮条 | 200 |
| 8.2.2. | scHoriz(...) | 画一水平文本亮条 | 201 |
| 8.2.2 | scsaveScrn(...) | 保存屏幕映象 | 188 |
| 8.2.2. | screstScrn(...) | 恢复以前保存的屏幕 | 188 |

| | | | |
|---|---|---|---|
| 8.2.3. | vrdChar(...) | 从屏幕读一个字符和属性值 | 189 |
| 8.2.3. | vdChar(...) | 往屏幕写一个字符和属性值 | 190 |
| 8.2.3. | vdChr(...) | 往屏幕写一个字符 | 190 |
| 8.2.3. | vdWrite(...) | 往屏幕写字符串 | 191 |
| 8.2.3. | vdAttr(...) | 仅仅改变屏幕属性 | 193 |
| 8.2.3. | vdVert(...) | 画一垂直文本亮条 | 201 |
| 8.2.3. | vdHoriz(...) | 画一水平文本亮条 | 202 |
| 8.2.3. | saveScrn(...) | 保存屏幕映象 | 194 |
| 8.2.3. | restScrn(...) | 恢复前面保存的屏幕 | 194 |
| 8.3. | onSound(...) | 使扬声器按规定的频率振动 | 203 |
| 8.3. | offSound(...) | 使扬声器停止振动 | 208 |
| 8.3. | Delay(...) | 发声延时 | 204 |
| 8.3. | Beep(...) | 发出一种声音 | 205 |
| 8.3. | Bleep(...) | 发出一种声音 | 205 |
| 8.4. | gtKey(...) | 取按下键的 16 位码 | 205 |
| 8.4. | gtChar(...) | 取 8 位字符值 | 206 |
| 8.4. | gtScan(...) | 取 8 位扫描码 | 207 |
| 8.4. | gtKBflag(...) | 取键盘标识状态 | 208 |
| 8.4. | gtKBstat(...) | 取键盘等待状态 | 194 |
| 8.4. | prompt(...) | 从键盘取字符串 | 195 |
| 8.4. | inpunm(...) | 从键盘取数据串 | 197 |
| 8.5. | setRect(...) | 设置 RECT 结构 | 198 |
| 8.5. | sizeRect(...) | 定义矩形屏幕区域 | 199 |
| 8.5. | clrRect(...) | 清除屏幕的矩形区域 | 209 |
| 8.5. | fillRect(...) | 按属性填充矩形块 | 209 |
| 8.5. | shftRect(...) | 移动 RECT 的坐标 | 209 |
| 8.5. | expdRect(...) | 扩展 RECT 的坐标 | 210 |
| 8.5. | boxRect(...) | 清除矩形块且画出边框 | 210 |
| 8.5. | saveRect(...) | 存储矩形屏幕区域的内容 | 212 |
| 8.5. | restRect(...) | 恢复前面保存的矩形屏幕区域内容 | 212 |
| 8.6. | wvdAttr(...) | 给窗口屏幕设置属性字节 | 213 |
| 8.6. | wvdChar(...) | 往窗口中写标记 | 213 |
| 8.6. | wvdHoriz(...) | 往窗口中写水平条 | 213 |
| 8.6. | wvdVert(...) | 往窗口中写垂直条 | 213 |
| 8.6. | wvdWrite(...) | 往窗口中写串 | 213 |
| 8.6. | rdlmg(...) | (内部函数) | 213 |
| 8.6. | wrlmg(...) | (内部函数) | 213 |
| 8.6. | rdWind(...) | (内部函数) | 213 |

| | | | |
|---|---|---|---|
| 8.6. | strWind(...) | 第一次显示窗口 | 213 |
| 8.6. | setAttr(...) | 置窗口缺省值 | 213 |
| 8.6. | setWind(...) | 置窗口尺寸 | 213 |
| 8.6. | sizeImg(...) | (内部函数) | 213 |
| 8.6. | remvWind(...) | 从屏幕上去掉窗口 | 213 |
| 8.6. | dsyWind(...) | 破坏窗口结构 | 213 |
| 8.6. | setBord(...) | 置窗口边框类型 | 213 |
| 8.6. | setTitle(...) | 置窗口标题 | 213 |
| 8.6. | dispWind(...) | 显示窗口(第二次) | 213 |
| 8.6. | wrBox(...) | (内部函数) | 213 |
| 8.6. | wrWind(...) | (内部函数) | 213 |
| 8.7. | msinit(...) | 初始化鼠标 | 219 |
| 8.7. | mson(...) | 打开鼠标 | 220 |
| 8.7. | msoff(...) | 关闭鼠标 | 220 |
| 8.7. | msstat(...) | 取鼠标状态 | 221 |


## 第二节 按名称字典排序索引


Beep(...) ........ 205
Bleep(...) ........ 205
boxRect(...) ........ 210
clrRect(...) ........ 209
Delay(...) ........ 204
dispWind(...) ........ 213
dsyWind(...) ........ 213
expdRect(...) ........ 210
fillRect(...) ........ 209
gtChar(...) ........ 206
gtCur(...) ........ 176
gtKBflg(...) ........ 208
gtKBstat(...) ........ 194
gtKey(...) ........ 205
gtSCan(...) ........ 207
inpnum(...) ........ 197
mkAttr(...) ........ 182
mkToken(...) ........ 183
msinit(...) ........ 219
msoff(...) ........ 220

mson(...) …… 220
msstat(...) …… 220
mvCur(...) …… 176
offCur(...) …… 178
offSound(...) …… 203
onCur(...) …… 178
onSound(...) …… 203
prompt(...) …… 195
rCloc(...) …… 178
rdImg(...) …… 213
rdWind(...) …… 213
remvWind(...) …… 213
restRect(...) …… 212
restScrn(...) …… 194
rmvCur(...) …… 177
rsizeCur(...) …… 181
saveRect(...) …… 212
saveScrn(...) …… 194
scAttr(...) …… 187
scChar(...) …… 185
scChr(...) …… 187
scHoriz(...) …… 201
sCloc(...) …… 178
scrdChar(...) …… 184
screstScrn(...) …… 188
scrnClr(...) …… 184
scsaveScrn(...) …… 188
scToken(...) …… 186
scVert(...) …… 200
SCWrite(...) …… 200
setAttr(...) …… 213
setBord(...) …… 213
setRect(...) …… 198
setTitle(...) …… 213
setWind(...) …… 213
shftRect(...) …… 209
sizeCur(...) …… 180
sizeimg(...) …… 213

sizeRect(. . . ) …… 199
ssizeCur(. . . ) …… 181
strWind(. . . ) …… 213
vdAttr(. . . ) …… 193
vdChar(. . . ) …… 189
vdChr(. . . ) …… 190
vdHoriz(. . . ) …… 202
vdVert(. . . ) …… 201
vdWrite(. . . ) …… 191
vrdChar(. . . ) …… 189
wrBox(. . . ) …… 213
wrImg(. . . ) …… 213
wrWind(. . . ) …… 213
wvdAttr(. . . ) …… 213
wvdChar(. . . ) …… 213
wvdHoriz(. . . ) …… 213
wvdVert(. . . ) …… 213
wvdWrite(. . . ) …… 213

## 第三节 源代码清单

GTCUR. C

```
///////////////////////////////////////
//GTCUR. C
//Description:
// Gets the current cursor
// Location on page 0
//Entry:
// int *row -> currrent row
// int *col -> current column
////////////////////////////////////////
#include <dos.h>
#include <tproto.h>

void
gtcur(int *row,int *col)
{
union REGS ir,or;
```

```
ir.h.bh = 0;
ir.h.bh = 3;
int386(0x10,&ir,&or);
*row = (int)or.h.dh;
*col = (int)or.h.dl;
}
```

MVCUR.ASM

```
////////////////////////////////////////
//MVCUR.ASM
//Description:
// Move cursor
//Entry:
// p1 short int row
// p2 short int col
////////////////////////////////////////
;declare 80386 code
      .386

;parameters
row       EQU [EBP+8]
column EQU [EBP+12]

;declare model & code segment
               DOSSEG

               .MODEL SMALL

               .CODE

;declare -mvCur public

               PUBLIC -mvCur

;begin procedure
-mvCur PROC
      push ebp                   ;set stack frame
      mov ebp,esp
```

```
    mov ah,0fh              ;get current vid page to bh
    int 10h                 ;via BIOS
    mov dh,row              ;parameter 1 —> dh
    mov dl,column           ;parameter 2 —> dl
    mov ah,2                ;move cursor
    int 10h                 ;via BIOS
    mov esp,ebp             ;restore
    pop ebp
    ret
-mvCur ENDP
    END
```

RMVCUR. C

```
//////////////////////////////////////
//RMVCUR. C
//
//Description:
// Move the cursor relative to the current
// cursor location
//
//Entry:
// int r-offset —> row offset
// int c-offset —> col offset
//////////////////////////////////////
#include <tproto. h>

void
rmvCur(int r-offset,int c-offset)
{
int row,column;
gtCur( * row,&column);
row += r-offest;
column +=c-offest;
mvCur(row,column);
}
```

SCLOC. C    RCLOC. C

```
////////////////////////////////////////////
//SCLOC. C
//RCLOC. C
//Description:
// Saves and restores the current cursor
// location
////////////////////////////////////////////
#include <dos.h>
#include <tproto.h>

static unsigned char c-row;
static unsigned char c-col;

//save the current cursor location

void sCloc()
{
union REGS ir,or;
ir.h.ah = 0x03;
ir.h.bh = 0x00;
int386(0x10,&ir,&or);
c-row = or.h.dh;
c-col = or.h.dl;
}

//restore previously saved cursor location

void rCloc()
{
    mvCur((int)c-row,(int)c-col);
}
```

ONCUR. C    OFFCUR. C

```
////////////////////////////////////////////
//ONCUR. C
//OFFCUR. C
//
//Description:
```

```
// Turn the cursor on
///////////////////////////////////////////////
#include <stdio.h>
#include <tproto.h>

#define C-MASK 0x2000

//extrens here

extern void s-shape(int);
extern int g-shape(void);

/////////////////////////////////////////
//oncur(...)
//Description:
// Turns the cursor on
/////////////////////////////////////////
void onCur()
{
s-shape(g-shape() & —C-MASK);
}

/////////////////////////////////////////
//offcur(...)
//Description:
// Turns the cursor off
/////////////////////////////////////////
void offCur()
{
s-shape(g-shape()|C-MASK);
}
```

S-SHAPE. ASM

```
;///////////////////////////////////////////////
;//S-SHAPE. ASM
;//
;//Description:
;// Resstore cursor shape
```

```
;///////////////////////////////////////////
;386 instructions
      .386

;first parameter

shape EQU [EBP+8]

;new style directives

      DOSSEG

      .MODEL SMALL

      .CODE

      PUBLIC -s-shape

-s-shape proc
      push ebp
      mov ebp,esp
      mov cx,shape
      mov ah,1
      int 10h
      mov esp,ebp
      pop ebp
      ret
-s-shape ENDP
      END
```

G-SHAPE. ASM

```
;///////////////////////////////////////////
;//
;//G-SHAPE. ASM
;//
;//Description:
;// Get cursor shape
;///////////////////////////////////////////
```

```
;386 instructions

    .386

;new style directives

    DOSSEG

    .MODEL SMALL

    .CODE

    PUBLIC -g-shape

-g-shape proc
    push ebp
    mov ebp,esp
    mov ah,3
    int 10h
    mov ax.cx
    mov esp,ebp
    pop ebp
    ret
-g-shape ENDP
    END
```

SIZECUR. C

```
//////////////////////////////////////////////
//SIZECUR. C
//
//Description:
// set the cursor size
//////////////////////////////////////////////
#include <dos.h>
#include <tproto.h>

void sizeCur(int start,int end)
{
```

```
union REGS ir,or;
char lsb,msb;

//cast 32 bit ints to 8 bit ints

lsb = (char)end;
msb = (char)start;

//set regs and call BIOS

ir.h.ah = 0x01;
ir.h.cl = lsb;
ir.h.ch = msb;
int386(0x10,&ir,&or);

}
```

SSIZECUR.C  RSIZECUR.C

```
//////////////////////////////////////////////
//SSIZECUR.C
//RSIZECUR.C
//Description:
// Save current cursor shape
//////////////////////////////////////////////
#include <dos.h>
#include <tproto.h>

static unsigned char -CL,-CH;

void ssizeCur()
{
union REGS ir,or;

//get cursor size

ir.h.ah = 0x03;
ir.h.bh = 0x00;
```

```
int386(0x10,&ir,&or);

//save to static unsigned chars

-CH = or.h.ch;
-CL = or.h.cl;
}

////////////////////////////////////////////////
//RSIZECUR(...)
//Description:
// Restore the proviously saved cursor shape
////////////////////////////////////////////////
void rsizeCur()
{
union REGS ir,or;

//set registers

ir.h.cl = -CL;
ir.h.ch = -CH;

//function set cursor size

ir.h.ah = 0x01;
int386(0x10,&ir,&0r);

}
```

MKATTR. ASM

```
;//////////////////////////////////////
;//MKATTR.ASM
;//Description
;// Make a screen attribute byte
;//Entry:
;// int foreground color
;// int background color
;// int intensity
```

```
;// int blink on off
;//Return:
;// 8 bit attribute in AL
;////////////////////////////////////

    .386

;parameters on stack
fore-c EQU[EBP+8]
back-c EQU[ebp+12]
inten-t EQU[EBP+16]
blink-t EQU[EBP+20]

;set up segment order,model and code
    DOSSEG
    .MODEL SMALL
    .CODE

;make procedure public
    PUBLIC -mkAttr

;mkAttr procedure
-mkAttr proc
    push EBP
    mov EBP,ESP
    xor AX,AX
    mov AX,back-c
    mov cl,4
    shl AX,CL
    or AX,fore-c
    or AX,inten-c
    or AX,blink-t
    mov ESP,EBP
    pop EBP
    ret
-mkAttr ENDP
    END
```

MKTOKEN. ASM

```
;///////////////////////////////////////
;//MKTOKEN. ASM
;//Description
;// Convers 8 bit character and 8 bit
;// attribute into 16bit screen token.
;//Entry:
;// parameter 1:8 bit character
;// parameter 1:8 bit character
;//Return:
;// 16 bit screen token in AX
;///////////////////////////////////////
     .386

;parameters on stack
char EQU[EBP+8]
attr EQU[ebp+12]

;set up segment order,model and code
     DOSSEG
     .MODEL SMALL
     .CODE

;make procedure public
PUBLIC -mkToken

;mkToken procedure
-mkToken proc
     push EBP
     mov EBP,ESP
     xor AX,AX
     mov AL,char
     mov AH,attr
     mov ESP,EBP
     pop EBP
     ret
-mkToken ENDP
     END
```

SCRNCLR. ASM

```
;//////////////////////////////////////
;//SCRNCLR. ASM
;//Description
;// Clear the screen using the normal
;// attribute (white on black —7) and
;// relocates the cursor to row 0,col 0
;///////////////////////////////////////
;Enable 386 assembly instructions
    .386

;parameters on stack
s-a EQU[EBP+8]

;Simplified MASM 5.0 + directives
    DOSSEG
    .MODEL SMALL
    .CODE

;make procedure public
PUBLIC -scrnClr

-scrnClr proc
    push EBP
    mov EBP,ESP
    xor AX,AX
    mov CX,AX
    mov DH,24
    mov DL,79
    mov BH,s-a
    mov AH,6
    int 10h
    mov AH,15
    int 10h
    mov DX,0
    MOV AH,2
    int 10h
```

```
        mov ESP,EBP
        pop EBP
        ret
-scrnClr ENDP
        END
```

SCRDCHAR. ASM

```
;/////////////////////////////////////////////
;//SCRDCHAR. ASM
;//Descripition:
;// Reads a screen token at the current
;// cursor location (moves the cursor to
;// that screen location
;//Entry:
;// parameter 1: row
;// parameter 2: column
;/////////////////////////////////////////////
        .386

;parameters on stack
row     EQU[EBP+8]
column EQU[EBP+12]
;set up segment order,model and code
        DOSSEG
        .MODEL SMALL
        .CODE
;make prcedure public
        PUBLIC -scrdChar
-scrdChar PROC

        push EBP
        mov EBP,ESP
        mov AH,0fh
        int 10h
        mov DH,row
        mov DL,column
        mov AH,02h
        int 10h
```

```
        mov AH,08h
        int 10h
        mov ESP,EBP
        pop EBP
        ret
_scrdChar ENDP
        END
```

SCCHAR. ASM

```
;//////////////////////////////////////////
;//SCCHAR. ASM
;//Descripition:
;// Write a token (char + attribute) to
;// location row &column. The cursor is
;// relocated at row/col location
;//////////////////////////////////////////
        .386

;define parameters and offset
prow      EQU BYTE PTR [EBP+8]
pcol      EQU BYTE PTR [EBP+12]
ptoken    EQU WORD PTR [EBP+16]
;Simplified MASM 5.0 + directives
        DOSSEG
        .MODEL SMALL
        .CODE
;make prcedure public
        PUBLIC _scChar
_scChar PROC

        push EBP
        mov EBP,ESP
        mov AH,02h
        xor BH,BH
        mov DH,prow
        mov DL,pcol
        int 10h
        mov ECX,1
```

```
    mov ax,ptoken
    mov BL,AH
    mov AH,09h
    int 10h
    mov ESP,EBP
    pop EBP
    ret
-scChar ENDP
    END
```

SCTOKEN. ASM

```
;//////////////////////////////////////////
;//SCTOKEN. ASM
;//Descripition:
;// WRITE a screen token at the current
;// cursor location
;//////////////////////////////////////////
    .386

;define parameters and offset
ptoken EQU WORD PTR[EBP+8]
;simplified MASM 5. 0+directive
    DOSSEG
    .MODEL SMALL
    .CODE
;make prcedure public
    PUBLIC -scToken
-scToken PROC
    push EBP
    mov EBP,ESP
    mov ECX,1
    mov AX,ptoken
    mov BL,AH
    mov AH,09h
    int 10h
    mov ESP,EBP
    pop EBP
    ret
```

```
-scToken ENDP
     END
```

SCCHR. ASM

```
;////////////////////////////
;// SCCHR. ASM
;////////////////////////////
.386

prow EQU BYTE PTR [EBP+8]
PCOL EQU BYTE PTR [EBP+12]
PTOKEN EQU WORD PTR [EBP+16]

DOSSEG
.MODEL SMALL
.CODE
PUBLIC -scChr

-scChr proc
      push  EBP
      mov   EBP, ESP
      mov   BH,0
      mov   DH,prow
      mov   DL,pcol
      mov   AH,2
      int   10H
      mov   AX,ptoken
      mov   AH,0AH
      mov   CX,1
      int   10H
      mov   ESP,EBP
      pop   EBP
      ret
-scChr ENDP
      END
```

SCATTR. C

```
;///////////////////////////////////////////////
;//SCATTR. C
;//Descripition:
;// Change a specified number of screen
;// charaters'  attribute while not
;// altering the displayed character.
;//Entry:
;// parameter 1: row
;// parameter 2: column
;// parameter 3: attributes to change
;// parameter 4: attribute
;///////////////////////////////////////////////
#include <tproto.h>
static local to file

void scAttr(short int row,short int col,short int length,short int attr)
{
        short int r,c,count,val;
        r=row;
        c=col;
        for (count=0;count<length;count++)
              vd1-buff[count]=(char)scrdChar(r,c++);
        scWrite(row,col,length,(char * )vdd1-buff,attr);
}
```

SCSAVESCRN. C , SCRESTSCRN. C

```
////////////////////////////////////////////////
//SCSAVESCRN. C , SCRESTSCRN. C
//Description:
// save screen and restore screen via BIOS. .
////////////////////////////////////////////////
#include <stdio.h>
#include <tproto.h>
#include <dos.h>

void scsaveScrn()
{
```

```
	int row,col;
	short int * ptr;
	ptr=sbuff;
	for (row=0;row<25;row++)
		for (col=0;col<80;col++)
			* ptr++ = scrdChar(row,col);
}

void screstScrn()
{
	union REGS ir,or;
	int row,col;
	short int * ptr;
	short int token;

	ptr = sbuff;
	for (row=0;row<25;row++)
		for (col=0;col<80;col++)
			scChar (row,col, * ptr++)
}
```

VRDCHAR. ASM

```
;////////////////////////////////////////////////
;//VRDCHAR. ASM
;//Descripition:
;// Read a screen token at the current cursor
;// location. This routine has been designed
;// to work with the PharLat DOS extender.
;//Entry:
;// parameter 1: row
;// parameter 2: column
;////////////////////////////////////////////////
.386

prow	EQU BYTE PTR [EBP+8]
pcol	EQU BYTE PTR [EBP+12]

DOSSEG
```

```
.MODEL SMALL
.CODE

PUBLIC -vrdChar
-vrdChar PROC
push EBP
mov EBP,ESP
push DI
push SI
mov CX,1ch
mov ES,CX
mov AX,0
mov AL,prow
mov BL,160
mul BL
mov CL,pcol
mov CH,0
shl CX,1
add AX,CX
mov DI,AX
mov AX,WORD PTR ES:[DI]
pop SI
pop DI
mov ESP,EBP
POP EBP
ret
-vrdChar ENDP
END
```

VDCHAR.ASM

```
;///////////////////////////////////////////////////////////////////
;//VDCHAR.ASM
;//Descripition:
;// Write a screen token to the location row & column.
;// The cursor is relocated at row/col location. This
;// routine has been designed to work with the PharLat
;// DOS extender.
;/////////////////////////////////////////////////////////
```

```
        .386

prow     EQU BYTE PTR [EBP+8]
pcol     EQU BYTE PTR [EBP+12]
ptoken   EQU WORD PTR [EBP+16]

        DOSSEG
        .MODEL SMALL
        .CODE

        PUBLIC -vdChar
-vdChar PROC
        push EBP
        mov EBP,ESP
        push DI
        push SI
        mov CX,1ch
        mov ES,CX
        mov AX,0
        mov AL,prow
        mov BL,160
        mul BL
        mov CL,pcol
        mov CH,0
        shl CX,1
        add AX,CX
        mov DI,AX
        mov AX,ptoken
        pop SI
        pop DI
        mov ESP,EBP
        pop EBP
        ret
-vdChar ENDP
        END
```

VDCHR.ASM

```
;/////////////////////////////////////////////////
;//VDCHR.ASM
;//Descripition:
;// Write a charater to location row & column.
;// the cursor is relocated at row.column.
;// This routine has been designed to work
;// with the PharLat DOS extender.
;//Entry:
;// parameter 1: row
;// parameter 2: column
;// parameter 3: character
;/////////////////////////////////////////////////
        .386

prow      EQU BYTE PTR [EBP+8]
pcol      EQU BYTE PTR [EBP+12]
pchar     EQU BYTE PTR [EBP+16]

        DOSSEG
        .MODEL SMALL
        .CODE

        PUBLIC -vdChr
-vdChr PROC
        push EBP
        mov EBP,ESP
        push DI
        push SI
        mov CX,1ch
        mov ES,CX
        mov AX,0
        mov AL,prow
        mov BL,160
        mul BL
        mov CL,pcol
        mov CH,0
        shl CX,1
        add AX,CX
```

```
        mov DI,AX
        mov AX,pchar
        pop SI
        pop DI
        mov ESP,EBP
        POP EBP
        ret
-vdChr ENDP
        END
```

VDWRITE. ASM

```
;///////////////////////////////////////////////
;//VDWRITE. ASM
;//Descripition:
;// Write a string of length bytes to the
;// screen using a specifid attribute.
;// This routine has been designed to work
;// with the PharLat DOS extender.
;//Entry:
;// parameter 1: row
;// parameter 2: column
;// parameter 3: number of string bytes to write
;// parameter 4: point to string
;// parameter 5: screen attribute
;///////////////////////////////////////////////
        .386

prow EQU BYTE PTR [EBP+8]
pcol EQU BYTE PTR [EBP+12]
plen EQU BYTE PTR [EBP+16]
pptr EQU BYTE PTR [EBP+20]
pattr EQU BYTE PTR [EBP+24]

        DOSSEG
        .MODEL SMALL
        .CODE

        PUBLIC -vdWrite
```

```
_vdWrite PROC
        push EBP
        mov EBP,ESP
        push DI
        push SI
        push DS
        push ES
        mov CX,1ch
        mov ES,CX
        mov SI,pptr
        mov AX,0
        mov AL,prow
        mov BL,160
        mul BL
        mov CL,pcol
        mov CH,0
        shl CX,1
        add AX,CX
        mov DI,AX
        cld
        mov ECX,plen
        mov AH,pattr
vdr1:
        lodsb
        stosb
        loop vdr1
done:
        pop ES
        pop DS
        pop SI
        pop DI
        mov ESP,EBP
        POP EBP
        ret
_vdWrite ENDP
        END
```

VDATTR. ASM

```
;///////////////////////////////////////////////////
;//VDATTR. ASM
;//Description:
;// Changes a specified number of screen characters'
;// attribute while not altering the displayed chracter.
;// This routine has been designed to work with the
;// PharLat DOS extender.
;//Entry:
;// parameter 1: row
;// parameter 2: column
;// parameter 3: attributes to change
;// parameter 4: attribute
;///////////////////////////////////////////////////
        .386

prow EQU BYTE PTR [EBP+8]
pcol EQU BYTE PTR [EBP+12]
plen EQU BYTE PTR [EBP+16]
pattr EQU BYTE PTR [EBP+20]

        DOSSEG
        .MODEL SMALL
        .CODE

        PUBLIC -vdAttr
-vdAttr PROC
        push EBP
        mov EBP,ESP
        push DI
        push SI
        mov CX,1ch
        mov ES,CX
        mov AX,0
        mov AL,prow
        mov BL,160
        mul BL
        mov CL,pcol
        mov CH,0
```

```
        shl CX,1
        add AX,CX
        mov DI,AX
        cld
        mov ECX,plen
        mov AL,pattr
vdr1:
        inc DI
        stosb
        loop vdr1
        pop SI
        pop DI
        mov ESP,EBP
        POP EBP
        ret
_vdAttr ENDP
        END
```

SAVESCRN.C , RESTSCRN.C

```
/////////////////////////////////////
//saveScrn.c , restScrn.c
//Desciption:
// save and restore screen via
// direct memory. (PharLap Version)
/////////////////////////////////////
#include <stdio.h>
#include <tproto.h>
static short int sbuff[2000]

void saveScrn()
{
        int row,col;
        short int *ptr;
        ptr = sbuff;
        for (row=0;row<25;row++)
            for (col=0;col<80;col++)
                        *ptr++ = vrdChar(row,col);
```

```
}

void restScrn()
{
        int row,col;
        short int *ptr;
        ptr = sbuff;
        for (row=0;row<25;row++)
            for (col=0;col<80;col++)
                    vdChar(row,col,*ptr);
}
```

GTKBSTAT. ASM

```
;///////////////////////////////////////////////
;//GTKBSTAT. ASM
;//Descripition:
;// Chechs to see if a keystroke is waiting.
;//Entry: (nothing)
;//Exit:
;// Returns
;// 0                 -> on no key waiting
;// 16 bit key code -> on key waiting
;///////////////////////////////////////////////
        .386

        DOSSEG
        .MODEL SMALL
        .CODE

        PUBLIC -gtKBstat
-gtKBstat PROC
        push EBP
        mov EBP,ESP
        mov AH,1
        int 16h
        jnz yeskey
        mov AX,0
        jmp keyexit
```

```
yeskey:
        mov AH,0
        int 16h
keyexit:
        mov ESP,EBP
        POP EBP     .
        ret
-gtKbstat ENDP
        END
```

PROMPT. C

```
////////////////////////////////////////////
//PROMPT. C
//Desciption:
// get a string from th ekeyboard and
// place it in a designed buffer.
//            ESC terminates prompt
// BACK space works
// ENTER terminate prompt
//Entry:
// parameter 1: char * to response buffer
// parameter 2: length of character field
//Exit:
// aTRUE --> prompt terminated by ENTER
// aFALSE --> prompt terminated by ESC
////////////////////////////////////////////
#include <string. h>
#include <tproto. h>
#include <stdio. h>

int prompt (char * response,int length)
{
        int key,exit;
        int row,column;
        int start,stop;
        int ret-val;
```

```
gtCur(&row,&column);

start = column;
stop = start + length;

exit = aFALSE;

do
{   mvCur(row,col);
    key = gtKey();
    switch (key)
  {         case ENTER: ret-val = aTRUE;
                        exit = aTRUE;
                        break;
           case ESCAPE:ret-val = aFALSE;
                        exit = aFALSE;
                        break;
           default:    key &= 0x00ff;
                        if (column < stop)
                        { mvCur(row,column);
                          scChr(row,column,key);
                          *response ++ = (char)key;
                          column++;
                        }
                        else
                        { beep();
                        }
                        if (key==aBS)
                        { if (column>start)
                           { column--;
                             response--;
                             mvCur(row,column);
                             scChr(row,column,' ');
                             *response = (char)aNULL;
                           }
                           else
                           { beep();
                           }
```

```
                        }
                        break;
            }
        } while(! exit);
respose = 0;
return(ret-val);
}
```

INPNUM. C

```
///////////////////////////////////////////////
//INPNUM. C
//Desciption:
// get aN intrger string from the keyboard
// and place it in a designed buffer.
//      ESC terminates prompt
// BACK space works
//      ENTER terminate prompt
//Entry:
// parameter 1: long * to value entered
// parameter 2: length of number field
//Exit:
// aTRUE -> prompt terminated by ENTER
// aFALSE -> prompt terminated by ESC
///////////////////////////////////////////////
#include <stdio. h>
#include <string. h>
#include <stdlib. h>
#include <tproto. h>

int inpnum (long * val,int length)
{
    int key,exit;
    int row,column;
    int start,stop;
    int ret-val;
    int response[50];
    char * cptr;
```

```
cptr = (char * )response;
memset(response,' \0' ,50);
gtCur(&row,&column);

start = column;
stop = start + length;

exit = aFALSE;

do
{   mvCur(row,col);
    key = gtKey();
    switch (key)
    {       case ENTER: ret-val = aTRUE;
                 * val = atol(response);
                 exit = aTRUE;
                 break;
           case ESCAPE:ret-val = aFALSE;
                 exit = aFALSE;
                 break;
        default: key &= 0x00ff;
        if (key>=0x20)&&(key<=0x7d)
        {
        if (column < stop)
        {   mvCur(row,column);
            if (isdigit(key)||key==' —' )
            {
            scChr(row,column,key);
             * cptr++ = (char)key;
            column++;
            }
            else
            {    beep();
            }
        }
       }
        if (key==aBS)
        {   if (column>start)
```

```
                    {   column--;
                        cptr--;
                        mvCur(row,column);
                        scChr(row,column,'  ');
                        *cptr = (char)aNULL;
                    }
                    else
                    {   beep();
                    }
                }
            break;
        }
    } while(! exit);
  return(ret-val);
}
```

SETRECT. C

```
////////////////////////////////////////////
//SETRECT. C
//Desciption:
// Allocates memory for a rect structure
// and sets the structure to parameters.
//Entry:
// paramter 1:RECT *
// PARAMETER 2:upper row int
// parameter 3:upper col int
// parameter 4:lower row int
// parameter 5:lower col int
//Exit:
// returns point to RECT
////////////////////////////////////////////
#include <malloc. h>
#include <tproto. h>
#include <string. h>

char b-rect-msg[]="NULL Returned in setRect";
RECT *setRect(RECT *R,int ur,int uc,int lr,setRect")
{     int size;
```

```
        R = (rect * )malloc (sizeof(RECT));
        r->ul-row = ur;
        r->ul-col = ur;
        r->lr-row = lr;
        r->lr-col = lc;

        size = sizeRect(R);
        R-image = (unsigned int * )calloc(size,sizeof(size ));

        return(R);
}
```

SIZERECT. C

```
//////////////////////////////////////////////////
//SIZERECT. C
//Desciption:
// Return the number of ints required to
// hold rectangle image
//Entry:
// paramter 1:point to RECT
//Exit:
// returns image size of rectangle in ints
//////////////////////////////////////////////////
#include <malloc. h>
#include <tproto. h>
#include <string. h>

unsigned int sizeRect(RECT * R)
{
        int height,widdth,size;

        height = R->lr-row - R->ul-row;
        with = R->lr-col - R->ul-col;

        size = (height+1) * (width +1)

        retur(size);
}
```

SCWRITE. C

```
///////////////////////////////////////////////
//SCWRITE. C
//
//Description:
// Writes a string of length bytes to
// to the screen using a specified attribute
///////////////////////////////////////////////
//include files here

#include <tproto.h>

void scWrite(short int r,short int c,short int l,char *p,short int a)
{
int cnt;
short int token;

for (cnt=0;cnt<l;cnt++)
     {

     mvCur(r,c++);
     token = mkToken(*p++,a);
     scToken(token);
     }
}
```

SCVERT. C

```
///////////////////////////////////////////////
//SCVERT. C
//
//Description:
// Draws in text mode a vertical line
// in the screen with a specified attribute
//
//Entry:
// parameter 1:row
// parameter 2:column
```

```
// parameter 3:height in rows
// parameter 4:attribue
////////////////////////////////////////////
#include <tproto.h>

void scVert(int row,int col,int length. int attr)
{
char buf179[80]={
      179,179,179,179,179,179,179,179,179,179
      179,179,179,179,179,179,179,179,179,179
      179,179,179,179,179,179,179,179,179,179
      179,179,179,179,179,179,179,179,179,179
      179,179,179,179,179,179,179,179,179,179
      179,179,179,179,179,179,179,179,179,179
      179,179.179.179,179,179,179,179,179,179
      179,179.179,179.179,179,179,179,179,179};
int ctr;

//write vertical column

for(ctr=row;ctr<row+length;ctr++)
    scWrite(ctr,col,1,buf179,attr);
}
```

SCHORIZ. C

```
////////////////////////////////////////////
//SCHORIZ. C
//
//Description:
// Write a horizontal bar in the text
// screen using a specified attribute
//
//Entry:
// parameter 1:row
// parameter 2:column
// parameter 3:width of bar in bytes
// parameter 4:attribue
```

```
/////////////////////////////////////////////
#include <tproto. h>

void scHorize(int row,int col,int length. int attr)
{
char buf196[80]={
      196,196,196,196,196,196,196,196,196,196
      196,196,196,196,196,196,196,196,196,196
      196,196,196,196,196,196,196,196,196,196
      196,196,196,196,196,196,196,196,196,196
      196,196,196,196,196,196,196,196,196,196
      196,196,196,196,196,196,196,196,196,196
      196,196,196,196,196,196,196,196,196,196
      196,196,196,196,196,196,196,196,196,196};

//write horizontal bar

     scWrite(ctr,col,length,buf196,attr);

}
```

VDVERT. C

```
/////////////////////////////////////////////
//VDVERT. C
//
//Description:
// Draws in text mode a vertical line
// in the screen with a specified
// attribute. this routines has been
// designed to work with th Phar Lap
// DOS extender.
//
//Entry:
// parameter 1: row
// parameter 2: column
// parameter 3: height in rows
// parameter 4: attribute
/////////////////////////////////////////////
```

```
#include <tproto.h>

void vdVert(int row,int col,int length,int attr)
{
int ctr;

for (ctr=row;ctr<row+length;ctr++)
     vdChar(ctr,col,mkToken(179,attr));
}
```

VDHORIZ.C

```
/////////////////////////////////////////////
//VDHORIZ.C
//
//Description:
// Draws in text mode a horizontal bar
// in the screen with a specified
// attribute. this routines has been
// designed to work with th Phar Lap
// DOS extender.
//
//Entry:
// parameter 1: row
// parameter 2: column
// parameter 3: width in rows
// parameter 4: attribute
/////////////////////////////////////////////
#include <tproto.h>

void vdHoriz(int row,int col,int length,int attr)
{
int cnt;

for (cnt=row;cnt<row+length;cnt++)
     vdChar(row,cnt,mkToken(196,attr));
}
```

ONSOUND.ASM

```
//////////////////////////////////////////////
//ONSOUND. ASM
//
//Description:
// turn the sound output
// via the timer on at
// a specified rate
//
//Entry:
// parameter 1:rate
//
//Exit:
// (nothing)
//////////////////////////////////////////////
      .386

tone EQU [ebp+8]

      DOSSEG
      .MODEL SMALL

      .CODE
      PUBLIC -onSound

-onSound PROC
     push ebp
     mov ebp,esp
     mov al,0b6h
     out 43h,al
     mov ax,tone
     out 42h,al
     mov al,ah
     out 42h,al
     in al,61h
     or al,3
     out 61h,3
     mov esp,ebp
     pop ebp
```

```
    ret
-onSound ENDP
    END
```

OFFSOUND. ASM

```
/////////////////////////////////////////////
//OFFSOUND. ASM
//
//Description:
// turn the sound output
// via the timer off
//
//Entry:
// (nothing)
//
//Exit:
// (nothing)
/////////////////////////////////////////////
    .386

    DOSSEG

    .MODEL SMALL

    .CODE

    PUBLIC -offSound

-offSound PROC
    in al,61h ;disable speaker
    and al,0fch ;output via timer
    out 61h,al
    ret
-offSound ENDP
    END
```

DELAY. C

```
/////////////////////////////////////////////
```

```
//DELAY.C
//
//Description:
// Simple delay.
//
//Entry:
// parameter 1:outer loop counter
// parameter 2:inner loop counter
//
//Exit:
// (nothing)
/////////////////////////////////////////////
#include <tproto.h>

void Delay(val1,val2)
int val1,val2;
{
int cnt1,cnt2;
for (cnt1=0;cnt1<val1;cnt1++)
      for (cnt2=0;cnt2<val2;cnt2++)
            cnt2=cnt2;
}
```

BEEP.C

```
/////////////////////////////////////////////
//BEEP.C
//
//Description:
// Produce a constant tone from
// the speaker
/////////////////////////////////////////////
#include <tproto.h>

void beep()
{
short int cnt;
onSound(500);
for(cnt=0;cnt<200;cnt++)
```

```
    Delay(100,20);
offSound();
}
```

BLEEP. C

```
//////////////////////////////////////////////
//BLEEP. C
//
//Description:
// Sweeping tone sound.
//////////////////////////////////////////////
#include <tproto. h>

void bleep()
{
int count;
for (count=1000;count>10;count --=5)
    {
    onSound(count);
    Delay(100,120);
offSOund();
}
```

GTKEY. ASM

```
;//////////////////////////////////////////////
;//GTKEY. ASM
;//
;//Description:
;// Wait for a key press. When a key is
;// pressed the key' s character code is
;// placed in al register and the key' s
;// scan code is placed in the ah register
;//
;//Entry:
;// (nothing)
;//
;//Exit:
;// al —> key' s character code
```

```
;// ah --> key' s scan code
;////////////////////////////////////////////
     .386

     DOSSEG

     .MODEL SMALL

     .CODE

     PUBLIC -gtKey

-gtKey PROC
     push ebp
     mov ebp,esp
     mov ah,0
     int 16h
     mov esp,ebp
     pop ebp
     ret
-gtKey ENDP
     END
```

GTCHAR. ASM

```
;////////////////////////////////////////////
;//GTCKAR. ASM
;//
;//Description:
;// Wait for a key press. When a key is
;// pressed the key' s character code is
;// placed in AL register . AH holds 0
;//
;//Entry:
;// (nothing)
;//
;//Exit:
;// al --> key' s character code
;////////////////////////////////////////////
```

```
    .386
    DOSSEG
    .MODEL SMALL
    .CODE
    PUBLIC -gtChar
-gtChar PROC
    push ebp
    mov ebp,esp
    mov ah,0
    int 16h
    mov ah,0
    mov esp,ebp
    pop ebp
    ret
-gtChar ENDP
    END
```

GTSCAN. ASM

```
;//////////////////////////////////////////////
;//GTSCAN. ASM
;//
;//Description:
;// Wait for a key press. When a key is
;// pressed the key's SCAN code is
;// placed in AL register . AH holds 0
;//
;//Entry:
;// (nothing)
;//
;//Exit:
;// al —> key's SCAN code
;//////////////////////////////////////////////
    .386
```

```
        DOSSEG

        . MODEL SMALL

        . CODE

        PUBLIC -gtScan

-gtScan PROC
        push ebp
        mov ebp,esp
        mov ah,0
        int 16h
        MOV AL,AH
        mov ah,0
        mov esp,ebp
        pop ebp
        ret
-gtScan ENDP
        END
```

GTKBFIAG. ASM

```
;////////////////////////////////////////////////
;//GTKBFLAG. ASM
;//
;//Description:
;// Return the keyboard status flags
;//
;// Bit                     key
;// 7 6 5 4 3 2 1 0         Pressed
;// 0 0 0 0 0 0 0 1         Right shift
;// 0 0 0 0 0 0 1 0         Left shift
;// 0 0 0 0 0 1 0 0         Control
;// 0 0 0 0 1 0 0 0         Alt
;// 0 0 0 1 0 0 0 0         Scroll on
;// 0 0 1 0 0 0 0 0         Num on
;// 0 1 0 0 0 0 0 0         Caps on
```

```
;// 1 0 0 0 0 0 0 0     Insert toggled
;//
;//Entry:
;// (nothing)
;//
;//Exit:
;// al --> keyboard status flags
;///////////////////////////////////////////////
      .386

      DOSSEG

      .MODEL SMALL

      .CODE

      PUBLIC -gtKBflag

-gtKBflag PROC
      push ebp
      mov ebp,esp
      mov ah,2
      int 16h
      mov ah,0
      mov esp,ebp
      pop ebp
      ret
-gtKBflag ENDP
      END
```

CLRRECT. C

```
////////////////////////////////////////
//CLRRECT. C
//
//Description:
// clears the rectangle using the
// normal (7) screen attribute
////////////////////////////////////////
```

```
#include <tproto.h>

void clrRect(RECT *R)
{
// use the BIOS to clear the rectangle

aclrrect(R->ul-row,R->ul-col,R->lr-row-1,R->lr-col-1,7);
}
```

FILLRECT. C

```
////////////////////////////////////////
//FILLRECT.C
//
//Description:
// clears the rectangle using the
// passed attribute screen attribute
////////////////////////////////////////
#include <tproto.h>

void fillRect(RECT *R,int attr)
{
// use the BIOS to clear the rectangle

aclrrect(R->ul-row,R->ul-col,R->lr-row-1,R->lr-col-1,attr);
}
```

SHFTRECT. C

```
///////////////////////////////////////////
//SHFTRECT.C
//
//Description:
// Shifts the row and column
// location of a rectangle . the size
// of the rectangle remains constant.
//
//Entry:
// parameter 1: pointer to RECT structure
// parameter 2: shift value for row loc.
```

```
// parameter 3: shift value for column loc.
//
/////////////////////////////////////////////
include <tproto. h>

void shftRect(RECT  * R,int row-s,int col-s)
{
//adjust row values in structure

R->ul-row+=row-s;
R->lr-row+=row-s;

//adjust column values in structure
R->ul-row+=col-s;
R->lr-row+=col-s;

}
```

EXPDRECT. C

```
/////////////////////////////////////////////////////////
//EXPDRECT. C
//
//Description:
// shift the row and column location of the rectangle so it
// expands. the centre of the rectangle remains constant.
//
//Entry:
// parameter1:point to rect structure
// parameter2:expand value for row.
// parameter3:expand value for column loc.
/////////////////////////////////////////////////////////
include<tproto. h>

void expdRect (RECT  * R,int row-s,int col-s)
{
            //adjust row values in structure

            R->ul-row  -=row-s;
```

```
        R->lr-row +=row-s;

        //adjust row values in structure

        R->ul-col -=col-s;
        R->lr-col +=col-s;
}
```

BOXRECT. C

```
///////////////////////////////////////////
//BOXRECT. C
//Desciption:
// Draws a box around a RECT structure
//Entry:
// parameter 1: RECT *
// parameter 2: box type
// parameter 3:attribute
///////////////////////////////////////////
#include <stdio. h>
#include <tproto. h>

static char xwb-blamk[80] =
    {  32,32,32,32,32,32,32,32,
       32,32,32,32,32,32,32,32,
       32,32,32,32,32,32,32,32,
       32,32,32,32,32,32,32,32,
       32,32,32,32,32,32,32,32,
       32,32,32,32,32,32,32,32,
       32,32,32,32,32,32,32,32,
       32,32,32,32,32,32,32,32,
       32,32,32,32,32,32,32,32,
       32,32,32,32,32,32,32,32
    };

void boxRect(RECT *R,int boxer,int attr1)
{      int row,column;
       int token;
```

```
int top-bot,left-right,ul,ur,ll,lr;
switch(boxer)
{   case 1: top-bot = 196;
            left-right =186;
            ul=214;
            ur=183;
            ll=211;
            lr=189;
            break;
    case 2: top-bot = 205;
            left-right =179;
            ul=213;
            ur=184;
            ll=212;
            lr=190;
            break;
    case 3: top-bot = 205;
            left-right =186;
            ul=201;
            ur=187;
            ll=200;
            lr=188;
            break;
    case 4: top-bot = 196;
            left-right =179;
            ul=218;
            ur=191;
            ll=192;
            lr=217;
            break;
}

token = mkToken((int)' ' ,attr1);
for (row=R->ul-row;row<R->lr-row;row++)
    vdWrite(row,R->ul-col,R->lr-col - R->ul-col,xwb-blank,attr1);
for (column=R->ul-col;column<R->lr-col-1;++column)
{   vdChar(R->ul-row,column,mkToken(top-bot,attr1));
    vdChar(R->lr-row-1,column,mkToken(top-bot,attr1));
```

```
        }
        for (row=R->ul-row;row<R->lr-row-1;++row)
        {   vdChar(row,R->ul-col,mkToken(left-right,attr1));
            vdChar(row,R->lr-col-1,mkToken(left-right,attr1));
        }
        vdChar(R->ul-row,R->ul-col,mkToken(ul,attr1));
        vdChar(R->ul-row,R->lr-col-1,mkToken(ur,attr1));
        vdChar(R->lr-row-1,R->ul-col,mkToken(ll,attr1));
        vdChar(R->lr-row-1,R->lr-col-1,mkToken(lr,attr1));
}
```

SAVERECT. C

```
////////////////////////////////////////
//saveResct. c
//Desciption:
// Save screen image below rectangle
// to previously allocated buffer
//Entry:
// parameter 1: RECT *
////////////////////////////////////////
#include <tproto. h>
void saveRect(RECT *R)
{       unsigned int *iptr;
        register int row;
        register int column;
        iptr = (unsigned int *) R->image;
        for (row->ul-row;row<=R->lr-row;row++)
        {   for (column=R->ul-col;column<=R->lr-col;column)
             {   *iptr++ = vrdChar(row,column);
             }
        }
}
```

RESTRECT. C

```
////////////////////////////////////////
//restResct. c
//Desciption:
// Restore screen image below rectangle
```

```
// from previously allocated buffer
//Entry:
// parameter 1: RECT *
////////////////////////////////////////
#include <tproto.h>
void restRect(RECT *R)
{       unsigned int *iptr;
        register int row;
        register int column;
        iptr = (unsigned int *) R->image;
        for (row->ul-row;row<=R->lr-row;row++)
        {  for (column=R->ul-col;column<=R->lr-col;column)
              {  vdChar(row,column, *iptr++);
              }
        }
}
```

WIND. C

```
///////////////////////////
//WIND. C
//
//Description;
//This file contains all the fumctions
//for the WINDOW routines.
//
//wvdattr(...)        Set Attributes for Window Screen Bytes
//wdchar(...)         Write Token to Window
//wdhoriz(...)        Write Horizontal Line to Window
//wdvert(...)         Write Vertical Bar to Window
//wdwrite(...)        Write String to Window
//rding(...)          (Internal Function)
//wring(...)          (Internal Function)
//rdwind(...)         (Internal Function)
//wring(...)          (Internal Function)
//rdwind(...)         (Internal Function)
//strwind(...)        Display Window First time
//setAttr(...)        Set window default attribute
//setWind(...)        Set window dimensions
```

```
//sizeImg(...)        (Internal Function)
//remvWind(...)       Remove window from screen
//dsyWind(...)        Destroy window structure
//setBord(...)        Set window border type
//setTitle(...)       Set window title
//dispWind(...)       Display window (second time on)
//wrBox(...)          (Internal Function)
//wrWind(...)         (Internal Function)
/////////////////////////////////////////////////////////
#include <stdio.h>
#include <dos.h>
#include <malloc.h>
#include <tproto.h>

extern VIDEO *crt;
extern unsigned int SCRNSEG;
extern int VID-PORT;
extern int SPARKLE-FLAG;

///////////////////////////////////
//      wvdAttr(...)
///////////////////////////////////
void wvdAttr(WIND *R,int row,int col,int length,char attr)
{       row += R->ul-row;
        col += R->ul-col;
        vdAttr(row,col,length,attr);
}

///////////////////////////////////
//      wvdChar(...)
///////////////////////////////////
void wvdChar(WIND *R,int row,int col,int token)
{       int cval,aval;
        row += R->ul-row;
        col += R->ul-col;
        vdChar(row,col,token);
}
```

```
/////////////////////////////////////
//       wvdHoriz(... )
/////////////////////////////////////
void wvdHoriz(WIND  * W,int row,int col int number,char attr)
{       int stop,col-start,token;
        row + = W->ul-row;
        col + = W->ul-col;
        token = mkToken(196,attr);
        stop = col + number;
        for (col-start=col;col start<stop;col-start++)
            vdChar(row,col-start,token);
}

/////////////////////////////////////
//       wvdVert(... )
/////////////////////////////////////
void wvdVert(WIND  * W,int row,int col,int number,char attr)
{       int stop,row-start,token;
        row + = W->ul-row;
        col + = W->ul-coi;
        token = mkToken(179,attr);
        stop = row + number;
        for (row-start=row;row-start<stop;row-start++)
            vdChar(row-start,col,token);
}

/////////////////////////////////////
//       wvdWrite(... )
/////////////////////////////////////
void wvdWrite(WIND  * W,int row,int col,int length,char  * str,char attr)
{       row + = W->ul-row;
        col + = W->ul-col;
        vdWrite(row,col,length,str,attr);
}

/////////////////////////////////////
// rdImg
/////////////////////////////////////
```

```
void rdImg(WIND *R)
{	int row,col;
	unsigned int far *buf-ptr;
	buf-ptr=(unsigned int far *)R->img-ptr;
	for (row=R->ul-row;row<=R->lr-row;row++)
		for (col=R->ul-col;col<=R->lr-col;col++)
			*buf-ptr++ = vrdChar(row,col);
}

//////////////////////////////////
// wrImg(...)
//////////////////////////////////
void wrImg(WIND *R)
{	int row,col;
	unsigned int far *img-ptr;
	img-ptr = (unsigned int far *) R->img-ptr;
	for (row=R->ul-row;rwo<=R->lr-row;row)
	{	vdChar(row,col, *img-ptr);
		img-ptr++;
	}
}

///////////////////////////////////
// rdWind
///////////////////////////////////
void rdWind(WIND *W)
{	int row,col;
	unsigned int far *buf-ptr;
	buf-ptr=(unsigned int far *)W->Wind-ptr;
	for (row=W->ul-row;row<=W->lr-row;row++)
		for (col=R->ul-col;col<=W->lr-col;col++)
		{	*buf-ptr = vrdChar(row,col);
		}
}

///////////////////////////////////
// strWind
///////////////////////////////////
```

```
void strWind(WIND * W)
{       char  * tptr;
        tptr = W->t-title;
        rdImg(W);
        wrBox(W);
        R->visible = 1;
        if (R->show-top)
            wvdWrite(W,0,W->top-offset,W->top-length,tptr,W->attr);

}

//////////////////////////////////
// setAttr
//////////////////////////////////
void setAttr(WIND  * W,int attr)
{       W->attr = attr;
}

//////////////////////////////////
// setWind
//////////////////////////////////
WIND  * setWind(WIND  * W,int ul-row,int ul-col,int lr-row,int lr-col)
{       int temp;
        W = (WIND  * )malloc(sizeof(WIND) * 2);
        if (W==0)
        {   return (0);
        }
        W->ul-row = ul-row;
        W->ul-col = ul-col;
        W->lr-row = ul-row;
        W->lr-col = ul-col;
        W->img-size = sizeImg(W);
        W->img-ptr = (unsigned int far  * )malloc(W->img-size);
        if(W->img-ptr==0)
        {exit(0);
        }
        W->wind-ptr = (unsigneed int far  * )malloc(W->img-size);
        if (W->wind-ptr == 0 )
```

```
	{	exit(0);
	}
	W->visible = aFALSE;
	W->box-type = 0;
	W->attr = NORMAL;
	W->t-title = 0;
	W->b-title = 0;
	W->show-top = aFALSE;
	W->show-bot = aFALSE;
	return (R);
}

////////////////////////////////////
// sizeImg(...)
////////////////////////////////////
unsigned int sizeImg(WIND *W)
{	int height,width,size;
	height = W->lr-row - W->ul-row;
	width = W->lr-col - W->ul-col;
	++height;
	++width;
	size =height * width;
	size *=4;
	return (size);
}

////////////////////////////////////
// remvWind(...)
////////////////////////////////////
void remvWind(WIND *W)
{	if (W->visible)
	{	rdWind(W);
		wrImg(W);
		W->visible = 0;
	}
}

////////////////////////////////////
```

```
// dsyWind(...)
////////////////////////////////////
void dsyWind(WIND * W)
{       if (W->img-ptr != NULL)
            free ((char * )W-img-ptr);
        if (W->img-ptr != NULL)
            free ((char * )W-wind-ptr);
        if (W != NULL)
            free(W);
}

////////////////////////////////////
//      setBord(...)
////////////////////////////////////
void setBord(WIND * W,int type)
{       W->box-type = type;
}

////////////////////////////////////
//      setTitle
////////////////////////////////////
void setTitle(WIND * W,char * top)
{       W->top-length = strlen(top);
        W->top->offset = ((W->lr-col-W->ul-col)-W->top length)/2;
        W->top-offset += 1;
        W->t-title = (char * )malloc(W->top-length + 1);
        memset (W->t-title,'\0' ,W->top-length + 1);
        strcpy (W->t-title,top);
        W->show-top = aTRUE;
}

////////////////////////////////////
//      dispWind(...)
////////////////////////////////////
void dispWind(WIND * W)
{
  if (! W->visible)
  {     rdlmd(W);
```

```
    wrWind(W);
    W->visible = 1;

/////////////////////////////////////
//      wrBox(...)
/////////////////////////////////////
char wb-blank[80] = {
      32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32,
      32,32,32,32,32,32,32,32};
void wrBox(WIND * W)
{     register int row,col;
      int token;
      int top-bot,left-right,ul,ur,ll,lr;
      switch(W—. box-type)
      { case 1: top-bot = 196;
                left-right = 186;
                ul = 214;
                ur = 183;
                ll = 211;
                lr = 189;
                break;
        case 2: top-bot = 205;
                left-right = 179;
                ul = 213;
                ur = 184;
                ll = 212;
                lr = 190;
                break;
```

```
    case 3: top-bot = 205;
            left-right = 186;
            ul = 201;
            ur = 187;
            ll = 200;
            lr = 188;
            break;
    default:top-bot = 196;
            left-right = 179;
            ul = 218;
            ur = 191;
            ll = 192;
            lr = 217;
            break;
  }
  for (row=W->ul-row;row<W->lr-row;row++)
      vdWrite(row,W->ul-col,W->lr-col - W->ul->col,
             wb-blank,W->attr);
  for (col=W->ul-col;col<W->lr-col;++col)
  {   vdChar(W->ul-row,col,mkToken(top-bot,W->attr));
      vdChar(W->ul-row,col,mkToken(top-bot,W->attr));
  }
  for (row=W->ul-row;row<W->lr-row;++row)
  {   vdChar(row,W->ul-col,mkToken(left-right,W->attr));
      vdChar(row,W->ul-col,mkToken(left-right,W->attr));
  }
  vdChar(W->ul-row,W->ul-col,mkToken(ul,W->attr));
  vdChar(W->ul-row,W->lr-col,mkToken(ur,W->attr));
  vdChar(W->lr-row,W->ul-col,mkToken(ll,W->attr));
  vdChar(W->lr-row,W->lr-col,mkToken(lr,W->attr));
}

//////////////////////////////////////
//      wrWind
//////////////////////////////////////
void wrWind(WIND * W)
{   register int row,col;
    unsigned int * img-ptr;
```

```
img-ptr=(unsigned int *)W->wind-ptr;
for(row=W->ul-row;row<=W->lr-row;row++)
  for(col=W->ul-col;col<=W->lr-col;col++)
    Char(row,col,*img-ptr);
    img-ptr++;
    }
  }
}
```

MSINIT. ASM

```
;/////////////////////////////////////
;//        MSINIT. ASM
;/////////////////////////////////////
   .386
    DOSSEG
   .MODEL SMALL
   .CODE
   PUBLIC -msinit
-msinit PROC
        push EBP
        mov EBP,ESP
        mov AX,0
        int 33h
        cmp AX,0
        je nomouse
        mov AX,BX
        mov ESP,EBP
        pop EBP
        ret
-msinit ENDP
        END
```

MSON. ASM

```
;/////////////////////////////////////
;//        MSON. ASM
;/////////////////////////////////////
        .386
```

```
        DOSSEG
        . MODEL SMALL
        . CODE
        PUBLIC -mson
-mson PROC
        push EBP
        mov EBP,ESP
        mov AX,1  ;turn on mouse
        int 33h
        mov ESP,EBP
        pop EBP
        ret
-mson ENDP
        END
```

MSOFF. ASM

```
;///////////////////////////////////
;//        MSOFF. ASM
;///////////////////////////////////
        . 386
        DOSSEG
        . MODEL SMALL
        . CODE
        PUBLIC -msoff
-msoff PROC
        push EBP
        mov EBP,ESP
        mov AX,2  ; turn off mouse
        int 33h
        mov ESP,EBP
        pop EBP
        ret
-msoff ENDP
        END
```

MSSTAT. ASM

```
;///////////////////////////////////////
;//        MSSTAT.ASM
;///////////////////////////////////////
        .386
X equ [ebp+8]
Y equ [ebp+12]
        DOSSEG
        .MODEL SMALL
        .CODE
        PUBLIC -msstat
-msstat PROC
        push EBP
        mov EBP,ESP
        mov AX,3 ;get mouse postion and button status
        int 33h
        mov AX,BX
        mov BX,X
        mov WORD PTR [BX],CX
        mov BX,Y
        mov WORD PTR [BX],DX
        mov ESP,EBP
        pop EBP
        ret
-msstat ENDP
        END
```

Document generated by Anna’s Archive around 2023-2024 as part of the DuXiu collection (https://annas-blog.org/duxiu-exclusive.html).

Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):
{
  "before_pdg2pic_conversion": {
    "filename": "MTI5NTQxNjYuemlw",
    "filename_decoded": "12954166.zip",
    "filesize": 31122389,
    "md5": "2f1a47af88ab3733c9df61cd7fbbd0e1",
    "header_md5": "cca4e287bbcb9f6292aeac8e7d83c7db",
    "sha1": "aeb43a1bb263f3ce11d41af4cd14b96bee455f4b",
    "sha256": "5749b57bdb58752de7e2a90446506962a7c8140a81ef50f3791dfa9dd036e734",
    "crc32": 3604202394,
    "zip_password": "52gv",
    "uncompressed_size": 39173648,
    "pdg_dir_name": "_12954166",
    "pdg_main_pages_found": 231,
    "pdg_main_pages_max": 231,
    "total_pages": 239,
    "total_pixels": 1206848160
  },
  "after_pdg2pic_conversion": {
    "filename": "MTI5NTQxNjYuemlw",
    "filename_decoded": "12954166.zip",
    "filesize": 31192606,
    "md5": "236f3774e45046780733207677e5d1a1",
    "header_md5": "2b039cde20b1b7ca84d7f5f5e75b1ae0",
    "sha1": "931ab944be182f2ae1295a7ebb6ad621381842cb",
    "sha256": "fa46db0c8f72a9744939e503b204078e119fc417c62d89126b10223e9303aa10",
    "crc32": 2717527340,
    "zip_password": "",
    "uncompressed_size": 39168850,
    "pdg_dir_name": "",
    "pdg_main_pages_found": 231,
    "pdg_main_pages_max": 231,
    "total_pages": 239,
    "total_pixels": 1526120160
  },
  "pdf_generation_missing_pages": false
}